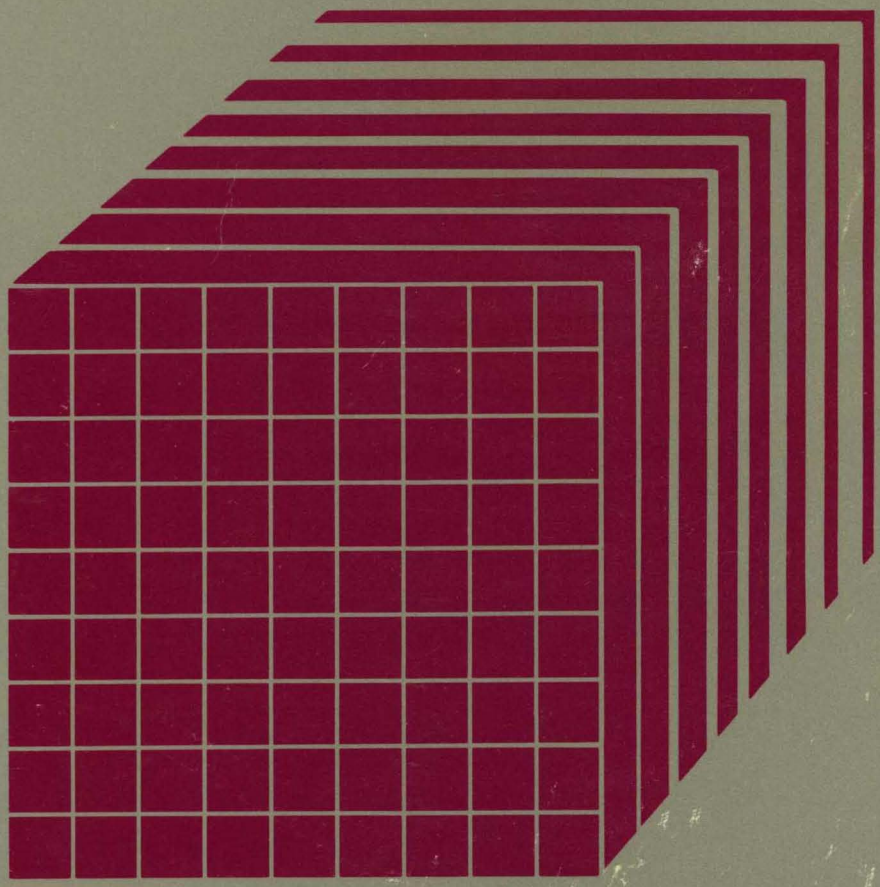
IBM

Virtual Machine/
System Product

CMS User's Guide

Release 3

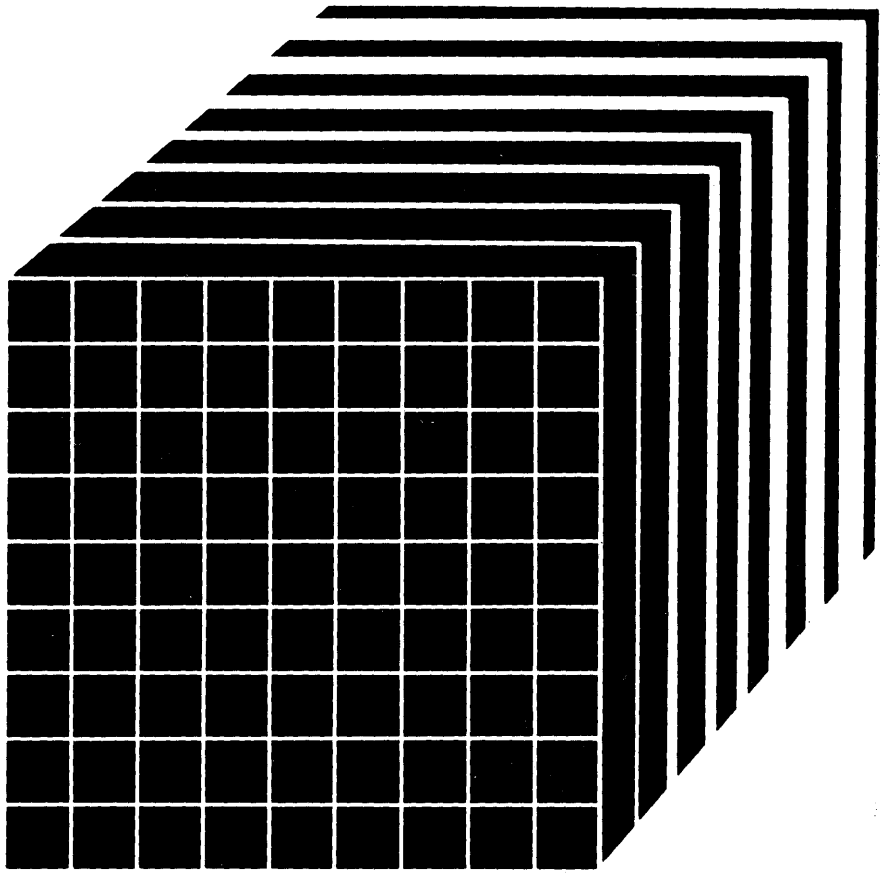


IBM

Virtual Machine/ System Product

CMS User's Guide

Release 3



Third Edition (September 1983)

This edition, SC19-6210-2, is a major revision of SC19-6210-1, and applies to Release 3 of the IBM Virtual Machine/System Product, (VM/SP), program number 5664-167, and to all subsequent releases and modifications until otherwise indicated in new editions or Technical Newsletters. Changes are periodically made to the information contained herein; before using this publication in connection with the operation of IBM systems, consult the *IBM System/370 and 4300 Processors Bibliography,* GC20-0001, for the editions that are applicable and current.

Summary of Changes

For a list of changes, see page iii.

Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM program product in this publication is not intended to state or imply that only IBM's program products may be used. Any functionally equivalent program may be used instead.

Publications are not stocked at the address given below. Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Programming Publications, Dept. G60, P.O. Box 6, Endicott, NY, U.S.A. 13760. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Summary of Changes

Summary of Changes
for SC19-6210-2
for VM/SP Release 3

Reorganization of this manual

- "Part 1. Understanding CMS" contains a new chapter about communicating with other computer users. Documentation on the CMS editor has been moved to "Appendix A." "Chapter 4. What You Can Do with CMS Commands" provides an overview of the operations that you might need to perform and the commands that you can use to perform these operations.

- The chapters in "Part 2. Program Development Using CMS" appear in a new order within the part.

- "Part 3: Learning to Use EXECs" contains overviews of the three EXEC interpreters. Documentation concerning the CMS EXEC Facility is now in "Appendix B."

- The chapters in "Part 4. The HELP Facility" have been condensed to two chapters.

New Commands for Release 3 of VM/SP

The following CMS commands are new for this release.

CATCHECK - Allows CMS VSAM users to invoke the VSE/VSAM Catalog Check Service Aid to verify a complete catalog structure.

EXECOS - Resets the OS and VSAM environments under CMS without returning to the interactive environment.

EXECUPDT - Used to apply updates to a System Product interpreter source program and create an executable version of the program.

IMMCMD - Establishes or cancels Immediate commands from within an EXEC.

RESERVE - Allocates all available blocks of a 512-, 1K-, 2K-, or 4K-byte block formatted minidisk to a unique CMS file.

New Immediate Commands

HI - Halt interpretation terminates execution of all currently executing System Product interpreter or EXEC 2 EXECs without destroying the environment as HX would.

TE - Trace end stops all tracing of your System Product interpreter or EXEC 2 programs or macros.

TS - Trace Start starts tracing your System Product interpreter or EXEC 2 programs or macros.

New CMS Function

DISKID - Obtains information on the physical organization of RESERVEd minidisk.

New CMS Macro Instructions

ABNEXIT - Sets or clears ABEND exit routines.
IMMCMD - Delcares, clears, or queries Immediate commands.
WAITECB - Waits on an Event control block (ECB) or a list of ECBs.

New CP Command

PER - Monitor certain events in the user's virtual machine as they occur during program execution.

The System Product Interpreter
"Part 3. Learning to Use EXECs" describes the System Product interpreter and provides examples of writing programs in the Restructured Extended Executor (REXX) language used with the System Product interpreter.

The XEDIT PF Keys
Document changes to the XEDIT PF Keys in HELP files and in other files.

512-byte blocksize
Document support of the 512-byte blocksize for CMS formatted minidisks.

Miscellaneous
This major revision incorporates minor technical and editorial changes.

Summary of Changes
for SC19-6210-1
for VM/SP Release 2

New:
Document changes due to the restructuring of the CMS nucleus.

Support of IOCP and the enhanced ASCII is included.

The following commands and functions are new for Release 2: CMDCALL, DEFAULTS, EXECIO, FILELIST, GLOBALV, IDENTIFY, NAMEFIND, NAMES, NOTE, NUCXDROP, NUCXLOAD, NUCXMAP, PEEK, RDR, RDRLIST, READCARD, RECEIVE, SENDFILE, TELL, and NUCEXT. These commands are documented in the *VM/SP CMS Command and Macro Reference.*

Changed:
This major revision incorporates minor technical and editorial changes.

Preface

This publication is intended for the general CMS user. It contains information describing the interactive facilities of CMS, and includes examples showing you how to use CMS.

This publication contains four parts, plus appendixes.

"Part 1: Understanding CMS" contains sections that describe, in general terms, the CMS facilities and the CMS and CP commands that you can use to control your virtual machine. If you are an experienced programmer who has used interactive terminal systems before, you may be able to refer directly to the *VM/SP CMS Command and Macro Reference* to find specific details about CMS commands that are summarized in this part. Otherwise, you may need to refer to later sections of this publication to gain a broader background in using CMS.

The topics discussed in Part 1 are:

- Getting Acquainted with VM/SP
- VM/SP-CMS Environments and Mode Switching
- The CMS File System
- What You Can Do with CMS Commands
- Editing Your Files
- Using Real Printers, Punches, Readers and Tapes
- Communicating with Other Computer Users

"Part 2: Program Development Using CMS" is primarily for applications programmers who want to use CMS to develop and test OS and VSE programs under CMS.

The topics discussed in Part 2 are:

- Programming for the CMS Environment
- Developing OS Programs Under CMS
- Developing VSE Programs Under CMS
- Using Access Method Services and VSAM Under CMS and CMS/DOS
- Using the CMS Batch Facility
- Debugging Your Programs

"Part 3: Learning To Use EXECs" gives detailed information on creating EXEC procedures to use with CMS.

The topics discussed in Part 3 are:

- Introduction to the EXEC Processors
- Creating System Product Interpreter EXECs
- Creating a PROFILE EXEC
- Exchanging Data Between Programs Through the Stack
- CMS Commands Used With System Product Interpreter EXECs

"Part 4: The HELP Facility" contains descriptions and examples of the use of HELP facility format words in creating HELP description files.

- Using the HELP Facility
- Tailoring the HELP Facility

"Appendix A: Using the CMS Editor"

"Appendix B: Using the CMS EXEC Facility"

"Appendix C: Considerations for Line-mode Terminals" discusses aspects of VM/SP and CMS that are different or unique when you use these terminals.

"Appendix D: Summary of CMS Commands" lists the commands available in the CMS command environment.

"Appendix E: Summary of CP Commands" describes the CP command privilege classes and summarizes the commands available in the CP command environment.

"Appendix F: Sample Terminal Sessions."

"Glossary" lists and defines terms that are used in this manual.

Terminology

Some of the following terms are used, for convenience, throughout this publication:

- The term "CMS/DOS" refers to the functions of CMS that become available when you issue the command

  ```
  set dos on
  ```

  CMS/DOS is a part of the normal CMS system, and is not a separate system. Users who do not use CMS/DOS are sometimes referred to as OS users, since they use the OS simulation functions of CMS.

- The term "CMS files" refers exclusively to files that are in the fixed block format used by CMS file system commands. VSAM and OS data sets and VSE files are not compatible with the CMS file format, and cannot be manipulated using CMS file system commands.

- The terms "disk" and "virtual disk" are used interchangeably to indicate disks that are in your CMS virtual machine configuration. Where necessary, a distinction is made between CMS-formatted disks and disks in OS or VSE format.

- The term "3270" refers to a series of display devices, namely, the IBM 3275, 3276 Controller Display Station, and 3277, 3278, and 3279 Display Stations. A specific device type is used only when a distinction is required between device types.

Information about display terminal usage also applies to the IBM 3138, 3148, and 3158 Display Consoles when used in display mode, unless otherwise noted.

Any information pertaining to the IBM 3284 or 3286 Printer also pertains to the IBM 3287, 3288, and 3289 printers, unless otherwise noted.

- The term "3330" refers to the IBM 3330 Disk Storage Models 1, 2, and 11, the IBM 3333 Disk Storage and Control Models 1 and 11, and the IBM 3350 Direct Access Storage in 3330 compatibility mode.

- The term "2305" refers to the IBM 2305 Fixed Head Storage, Models 1 and 2.

- The term "3340" refers to the IBM 3340 Direct Access Storage Facility and the IBM 3344 Direct Access Storage.
- The term "3350" refers to the IBM 3350 Direct Access Storage device when used in native mode.
- Any information pertaining to the IBM 2741 terminal also applies to the IBM 3767 terminal, unless otherwise noted.
- The term "370x" refers to the 3704/3705 Communications Controllers.
- The term "3370" refers to the IBM 3370 Direct Access Storage Device.
- The term "3310" refers to the IBM 3310 Direct Access Storage Device.
- The term "FB-512" refers to the IBM 3370 and 3310 Direct Access Storage Devices.

For a glossary of VM/SP terms, see the *Virtual Machine/System Product Library Guide and Master Index*, GC19-6207.

SCRIPT/VS is a component of the IBM Document Composition Facility program product, which is available from IBM for a license fee. For additional information on SCRIPT/VS usage, see *Document Composition Facility: User's Guide*, SH20-9161.

Prerequisite Publications

Virtual Machine/System Product:

Introduction, GC19-6200

Terminal Reference, GC19-6206

A new user of CMS might refer to the *VM/SP: CMS Primer*, SC24-5236, for introductory tutorial information on using CMS.

Corequisite Publications

Virtual Machine/System Product:

CMS Command and Macro Reference, SC19-6209

CP Command Reference for General Users, SC19-6211

EXEC 2 Reference, SC24-5219

Operating Systems in a Virtual Machine, GC19-6212

System Messages and Codes, SC19-6204

System Product Editor Command and Macro Reference, SC24-5221

System Product Editor User's Guide, SC24-5220

System Product Interpreter Reference, SC24-5239

System Product Interpreter User's Guide, SC24-5238

Quick References

There are publications available as quick reference material when you use VM/SP and CMS. They are:

Virtual Machine/System Product:

Commands (General User), SX20-4401

Commands (Other than General User), SX20-4402

EXEC 2 Language, SX24-5124

Quick Guide for Users, SX20-4400

SP Editor Command Language, SX24-5122

System Product Interpreter Reference Summary, SX20-5126

Related VM/SP Publications

Additional descriptions of various CMS functions and commands that are normally used by system support personnel are described in the following publications:

Virtual Machine/System Product:

Installation Guide, SC24-5237

Operator's Guide, SC19-6202

Planning Guide and Reference, SC19-6201

System Programmer's Guide, SC19-6203

Information describing the CMS command CPEREP, a command used to generate output reports from VM/SP's error recording records, is contained in the *Virtual Machine/System Product OLTSEP and Error Recording Guide*, SC19-6205

Details on the use of OS/VS EREP operands, required to make use of CPEREP, are contained in the *OS/VS, DOS/VSE, VM/370 Environmental Recording, Editing, and Printing Program*, GC28-0772.

IPCS CMS commands are described in *IBM Virtual Machine Facility/370: Interactive Problem Control System (IPCS) User's Guide*, GC20-1823, and not in this publication.

Related Publications for OS Users

For information on OS/VS tape label processing, discussed with "Label Processing in OS Simulation" in this publication, refer to:

OS/VS1 Data Management Services Guide, GC26-3874

OS/VS2 Data Management Services Guide, GC26-3875

OS/VS2 Data Management Services Guide, GC26-3875

OS/VS Tape Labels, GC26-3795

Information on the linkage editor is contained in *OS/VS Linkage Editor and Loader,* GC26-3813.

Related Publications for VSAM and Access Method Services Users

CMS support of Access Method Services is based on VSE and VSE/VSAM. The control statements that you can use are described in *Using VSE/VSAM Command and Macros,* SC24-5144.

Error messages produced by the Access Method Services program, and return codes and reason codes, are listed in *VSE/VSAM Messages and Codes,* SC24-5146.

For a detailed description of VSE/VSAM macros and macro parameters, refer to the *VSE/AF Macro User's Guide,* SC24-5210

For information on OS/VS VSAM macros, refer to *OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide,* GC26-3818.

Information on formatting virtual minidisks using the Device Support Facility Program is found in the *Device Support Facilities User's Guide and Reference,* GC35-0033.

Related Publications for CMS/DOS Users

The CMS ESERV command invokes the VSE ESERV program, and uses, as input, the control statements that you would use in VSE. These control statements are described in *Guide to the DOS/VSE Assembler,* GC33-4024.

Linkage editor control statements, used when invoking the linkage editor under CMS/DOS, are described in *VSE/AF System Control Statements,* SC33-4024.

For information on DOS/VSE and CMS/DOS tape label processing, refer to the following publications:

VSE/AF Tape Labels, SC24-5212

VSE/AF Macro User's Guide, GC24-5211

For information about using DL/I in the CMS/DOS environment, see *DL/I DOS/VS Data Base Administration,* SH24-5011.

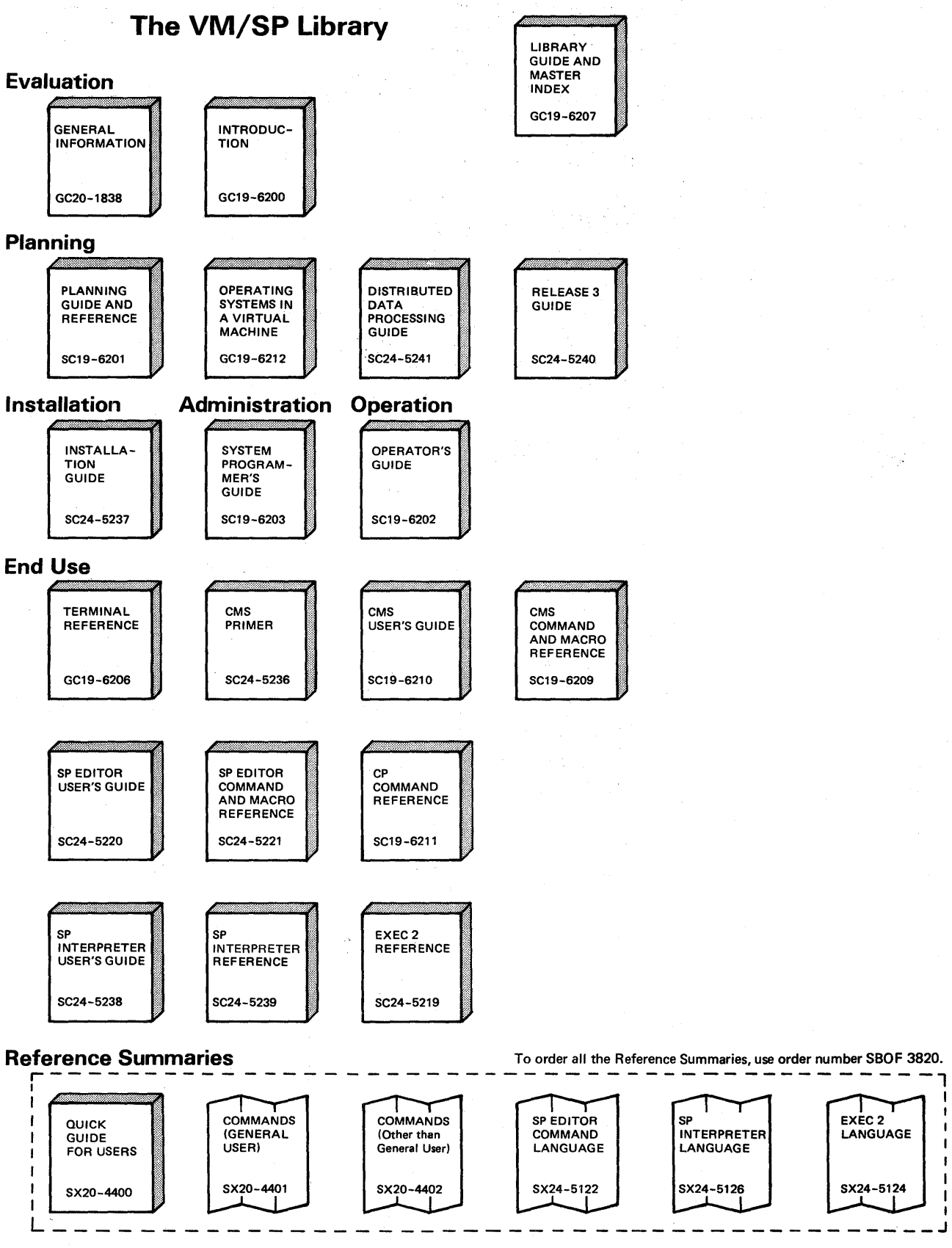


Figure 0-1 (Part 1 of 2). VM/SP Library Interrelationship of Publications

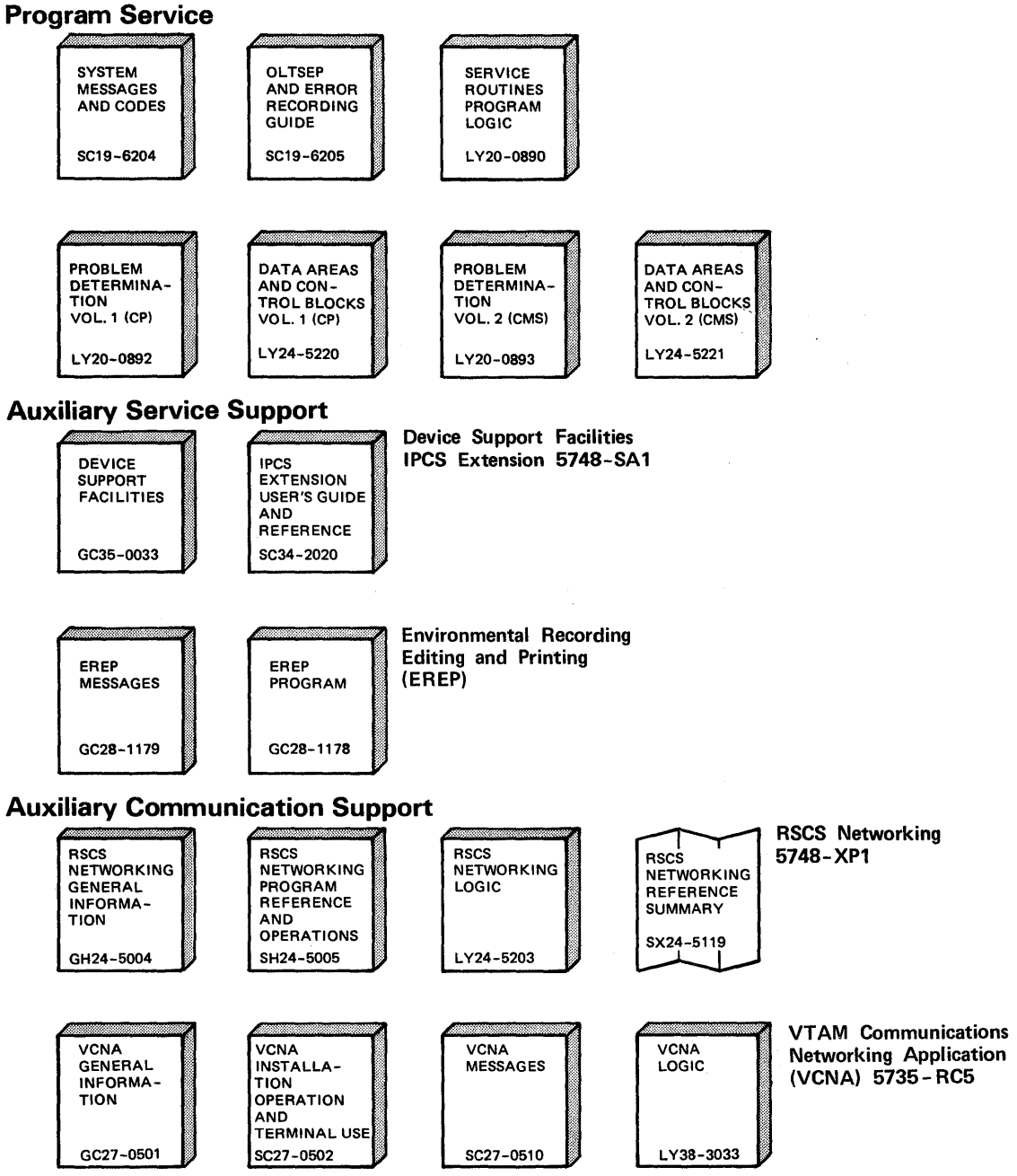


Figure 0-1 (Part 2 of 2). VM/SP Library Interrelationship of Publications

Contents

Figures

Part 1: Understanding CMS

Learning how to use CMS is not an end in itself: you have specific tasks to do, and you need to use the computer to perform them. CMS has been designed to make these tasks easier, but if you are unfamiliar with CMS, then the tasks may seem more difficult. The information contained in Part 1 of the *VM/SP CMS User's Guide* is organized to help you make the acquaintance of CMS quickly, so that it enhances, rather than impedes, the performance of your tasks.

Chapter 1, "Getting Acquainted with VM/SP" introduces you to VM/SP and its conversational component, CMS. It should help you to get a picture of how you, at a terminal, use and interact with the system.

During a terminal session, commands and requests that you enter are processed by different parts of the system. How and when you can communicate with these different programs, is described in Chapter 2, "VM/SP Environments and Mode Switching."

Almost every CMS command that you enter results in some kind of activity with a direct access storage device (DASD), known in CMS simply as a disk, or minidisk. Data and programs are stored on disks in what are called "files." Chapter 3, "The CMS File System" introduces you to the creation and handling of CMS files.

There are more than two hundred commands and subcommands comprising the VM/SP language. There are some that you may never need to use; there are others that you will use over and over again. Chapter 4, "What You Can Do with CMS Commands" contains a sampling of commands in various functional areas, to give you a general idea of the kinds of things you can do, and the commands available to help you do them.

Chapter 5, "Editing Your Files" contains some of the basic information you need to create and write a disk file directly from your terminal, or to correct or modify an existing CMS file.

Chapter 6, "Using Real Printers, Punches, Readers, and Tapes" discusses how to use tapes and punched cards in CMS, and how to use your virtual printer and punch to get real output.

Chapter 7, "Communicating with Other Computer Users" discusses the ways in which you can send information to other users and can receive information from them.

Chapter 1. Getting Acquainted with VM/SP

Virtual Machine/System Product (VM/SP) is a program product that controls "virtual machines." A virtual machine is the functional equivalent of a real computer that you control from your terminal, using a command language of verbs and nouns.

The command languages correspond to the components of VM/SP. CP controls the resources of the real machine; that is, the physical machine in your computer room; it also manages the communications among virtual machines, and between a virtual machine and the real system. CMS is the conversational operating system designed specifically to run under CP; it can simulate many of the functions of the OS and DOS operating systems, so that you can run many OS and DOS programs in a conversational environment.

Although this publication is concerned primarily with using CMS, it also contains examples of CP commands with which you, as a CMS user, should be familiar.

How You Communicate With VM/SP

When you are running your virtual machine under VM/SP, each command, or request for work, that you enter on your terminal is processed as it is entered; usually, you enter one command at a time and commands are processed in the order that you enter them.

You can enter CP commands from either the CP or CMS environment; but you cannot enter CMS commands while in the CP environment. The concept of "environments" in VM/SP is discussed in Chapter 2, "VM/SP Environments and Mode Switching."

After you have typed or keyed in the line you wish to enter, you press the Return or ENTER key on the keyboard. When you press this key, the line you have entered is passed to the command environment you want to have process it. If you press this key without entering any data, you have entered a "null line." Null lines sometimes have special meanings in VM/SP.

If you make a mistake entering a command line, VM/SP tells you what your mistake was, and you must enter the line again. The examples in this publication assume that the command lines are correctly entered.

You can enter commands using any combination of uppercase and lowercase characters; VM/SP translates your input to uppercase. Examples in this publication show all user-entered input lines in lowercase characters and system responses in uppercase characters.

The CP Command Language

You use CP commands to communicate with the control program. CP commands control the devices attached to your virtual machine and their characteristics.

For example, if you want to allocate additional disk space for a work area or if you want to increase the virtual address space assigned to your virtual machine, use the CP command DEFINE. CP takes care of the space allocation for you and then allows your virtual machine to use it.

Or if, for example, you are receiving printed output at your terminal and do not want to be interrupted by messages from other VM/SP users, you can use the CP command SET MSG OFF to refuse messages, because it is CP that handles communication among virtual machines. The CP QUERY SET command displays the status of the CP SET MSG function and other CP SET command functions.

Using CP commands, you can send messages to the system operator and to other users, or you can modify the configuration of devices in your virtual machine. CP commands are available to all virtual machines using VM/SP. You can invoke these commands when you are in the virtual machine environment using CMS (or some other operating system) in your virtual machine.

The CP commands and command privilege classes (not all commands are available to all users) are listed in Appendix E, "Summary of CP Commands." The CP Commands applicable to the average user are discussed in detail in the *VM/SP CP Command Reference for General Users*. The rest of the CP commands are discussed in *VM/SP Operator's Guide*. However, because many CP commands are used with CMS commands, some of the CP commands you will use most frequently are discussed in this publication, in the context of their usefulness for a CMS application. To aid you in distinguishing between CMS commands and CP commands, all CP commands used in examples in this publication are prefaced with "CP."

The CMS Command Language

The CMS command language allows you to create, modify, and debug problem or application programs and, in general, to manipulate data files.

Many OS language processors can be executed under CMS: the assembler, VS BASIC, OS FORTRAN, VS FORTRAN, OS/VS COBOL, and OS PL/I Optimizing and Checkout Compilers. In addition, the DOS/VS COBOL, DOS PL/I, VS APL, and DOS VS RPG II Program Products are supported. You can find a comprehensive list of language processors that can be executed under CMS and relevant publications in the *VM/SP Introduction*. CMS executes the assembler and the compilers when you invoke them with CMS commands. The ASSEMBLE command is used to present examples in this publication; the supported compiler commands are described in the appropriate DOS and OS program product documentation.

When you issue the XEDIT command, you invoke the System Product editor to create, modify, or manipulate CMS disk files. Once the VM/SP System Product editor has been invoked, you may execute XEDIT subcommands and use the System Product interpreter or EXEC 2 macro facility. When you invoke the EDIT command, the System Product editor places you in CMS (EDIT) migration mode. In this mode the you can use both EDIT and XEDIT subcommands to modify files. The System Product interpreter, CMS EXEC interpreter, and the EXEC 2 interpreter provide execution procedures consisting of CP and CMS commands; they also provide the conditional execution capability of a macro language. The DEBUG command gives you several program debugging subcommands.

Other CMS commands allow you to read cards from a virtual card reader, punch cards to a virtual card punch, and print records on a virtual printer. Many commands are provided to help you manipulate your virtual disks and files.

You use the HELP command to display at your terminal information on how to use CP commands and CMS commands, subcommands, and EXECs, and explanations

of CP and CMS messages. You can issue the HELP command when a brief explanation of syntax, a parameter, or function is sufficient, thereby avoiding interrupting your terminal session to refer to a manual.

Since you can invoke CP commands from within the CMS virtual machine environment, the CP and CMS command languages are, for practical purposes, a single, integrated command language for CMS users.

What You Must Know to Use VM/SP

Before you can use CP and CMS, you should know:

1. how to operate your terminal
2. your userid (user identification) and password.

The Terminal: Your Virtual Console

There are many types of terminals you can use as a VM/SP virtual console. Before you can conveniently use any of the commands and facilities described in this publication, you have to familiarize yourself with the terminal you are using. Generally, you can find information about the type of terminal you are using and how to use it with VM/SP in the *VM/SP Terminal Reference*. If your terminal is a 3767, you also need the *IBM 3767 Operator's Guide*.

In this publication, examples and usage notes assume that you are using a display terminal (such as a 3277). If you are using a typewriter style terminal (such as a 2741) consult Appendix C, "Considerations for Line Mode Terminals" for a discussion of special techniques that you can use to communicate with VM/SP.

Your Userid and Password: Keys into the System

Your userid is a symbol that identifies your virtual machine to VM/SP and allows you to gain access to the system. Your password is a symbol that functions as a protective device ensuring that only those allowed can use your virtual machine. The userid and password are usually defined by the system programmer for your installation.

Beginning Your Terminal Session

To establish contact with VM/SP, you switch the terminal device on and VM/SP responds with some form of the message:

```
VM/370 online
```

to let you know that VM/SP is running and that you can use it. If you do not receive the "VM/370 online" message, see the *VM/SP Terminal Reference* for specific directions. You can now press the ENTER key (or equivalent) on your terminal to clear the display. Now, enter your first command to identify yourself to VM/SP, the CP LOGON command. If your userid is TIGER, then you type:

```
cp logon tiger
```

and press the ENTER key. You only need to type L, because L is short for LOGON. Remember that throughout this publication all CP command will be preceded with "CP" so that you can distinguish them from CMS commands.

If VM/SP accepts your userid, it responds by asking you for your password:

```
ENTER PASSWORD:
```

Now, carefully type your password, and press the ENTER key. You may not see your password as you type it. This is a security measure and it prevents others from learning your password. If you receive the message, PASSWORD INCORRECT, you will have to start over, beginning with the CP LOGON command.

Getting Into CMS

After a successful logon, your next step is to load CMS in your virtual machine using the CP IPL command. IPL stands for Initial Program Load.

```
cp ipl cms
```

where "cms" is assumed to be the saved system name for your installation's CMS. VM/SP responds by displaying a message such as:

```
VM/SP CMS - 05/16/83  12:54
```

to indicate that the IPL command executed successfully. Press the ENTER key again. VM responds with a message, the last line, known as the ready message, which may look like this:

```
R; T=0.01/0.01 08:05:50
```

At this point you have IPL'ed CMS and you can now enter both CP and CMS commands.

Your userid may be set up for an automatic IPL, so that you receive a message, indicating that you are in the CMS command environment, without having to issue the IPL command.

> **Note:** If this is the first time you are using a new virtual disk assigned to you, you receive the message:

```
DMSACC112S 'A(191)' DEVICE ERROR
```

and you must "format" the disk, that is, prepare it for use with CMS files. See "Formatting Virtual Disks" below.

Ending Your Terminal Session

To end your terminal session, use the CP LOGOFF command. Enter:

```
cp logoff
```

and press the ENTER key, or just enter:

```
cp log
```

and press the ENTER key, because LOG is short for LOGOFF.

At times you may be running a long program under one userid and wish to use your terminal for some other work. Then, you can disconnect your terminal using the CP command DISCONN:

```
cp disconn

or

cp disconn hold
```

Your virtual machine continues to run, and is logged off the system when your program has finished executing. If you want to regain terminal control of your virtual machine after disconnecting, log on as you would to begin your terminal session. Your virtual machine is placed in the CP environment, and to resume its execution, you use the CP command BEGIN. You should not disconnect your virtual machine if a program requires an operator response, since the console read request cannot be satisfied.

Logon Procedure Summary

Remember to press the ENTER key after you type a command.

1. Enter CP LOGON userid
2. Enter your password when prompted
3. Enter IPL CMS
4. Press the ENTER key again.

Logoff Procedure Summary

1. Enter CP LOGOFF

Entering Commands

The IBM 3270 display terminal, commonly referred to as a 3270, functions somewhat differently from a typewriter-style terminal when you use it as a virtual machine console under VM/SP. Apart from the obvious difference in the way output is displayed, there are special techniques you can use with a 3270 that you cannot use on a 2741 or other typewriter terminals. Since the keyboard on a 3270 is never locked during the execution of a command or program, you can enter successive command lines without waiting for the completion of the previous command. This stacking function can be combined with the other methods of stacking lines, such as using the logical line end symbol (#) to stack several command lines. If you try to enter more lines than the terminal buffer can accommodate, however, you receive the status message NOT ACCEPTED and you must wait until the buffer is cleared before you can enter the line.

You will find, as you become accustomed to using a 3270, that the #CP function is very useful. The #CP function allows you to pass a command line to the control program immediately, bypassing any processing by the virtual machine (CMS). The #CP function can be used in any VM/SP environment, and you can enter it even when a program is executing. You do not have to interrupt a program's execution to enter a command line such as:

```
#cp display psw
```

to display the current PSW, or:

```
#cp spool printer class s
```

to spool your virtual printer.

RETRIEVE Function

One of the most common user difficulties is typing errors. The RETRIEVE function provides a convenient and time-saving method of correcting errors without retyping the entire input. You can use this function by defining a program function (PF) key for it, using a command such as:

```
#CP set pf12 retrieve
```

If you define a PF key for the RETRIEVE function, VM/SP remembers each input line entered at the terminal. When you press the PF key, VM/SP redisplays the latest input line in the input area, so that you can modify and re-enter the data. This allows you to correct errors, change your input, or repeatedly reissue a command.

VM/SP actually remembers several input lines. The number of lines remembered depends on the length of the lines; VM/SP remembers more short lines than long lines, but it can always remember at least one full input line. Duplicate input lines (lines that are the same as the previous input) are not remembered because it is not useful to remember the same line twice. For security reasons, input lines that are not displayed at the terminal, such as passwords, are never remembered.

When a RETRIEVE program function key is first pressed, VM/SP redisplays the latest input line. If a RETRIEVE key is pressed again, VM/SP displays the previous input line. As the key is pressed, VM/SP steps through the input lines displaying them one at a time. When VM/SP reaches the oldest line that it has remembered, it cycles back to the latest one again. When an input line is entered, VM/SP resets itself so that the RETRIEVE program function key starts with the latest input line.

> **Note:** For CP and CMS commands, there is a simple way to reset the RETRIEVE function to the latest input line: simply enter a single asterisk (*), which is treated as a comment by both CP and CMS. Then press the RETRIEVE program function key once to get the asterisk redisplayed, and a second time to get the previous input line redisplayed.

Setting Program Function Keys

If there are CP and CMS commands that you use frequently, you can set the program function (PF) keys on your terminal to execute them. Some examples of commands you might wish to catalog on PF keys are:

```
#CP DISPLAY PSW
#CP QUERY PRINTER ALL
QUERY SEARCH
```

To set functions keys 1, 2, and 3 to perform these command functions, enter:

```
cp set pf1 immed "#cp display psw
cp set pf2 immed "#cp query printer all
cp set pf3 immed query search
```

When you want to execute a #CP function with a PF key, or you want a PF key to execute a series of commands, you must use the logical escape symbol (") when you enter the SET command. For example:

```
cp set pf5 immed xedit test file"#bo"#input line"#file
```

sets the PF5 key as:

```
XEDIT TEST FILE#BO#INPUT LINE#FILE
```

The above examples use the IMMED operand of the SET command, which specifies that the function is performed as soon as you press the PF key. You can also set a key so that it is delayed; that is, the command or data line is placed in the user input area. Then, you must press the ENTER key to execute the command. You may modify the line before you enter it. This is the default setting (DELAY) for program function keys. For example, you might set a key as:

```
QUERY DISK X@
```

When you press this PF key, the command line is placed in the user input area, with the cursor positioned following the "@" logical character delete symbol; you can enter the mode letter of the disk you are querying before you press the ENTER key to execute the command. If you enter 'A', the resulting command as seen by CMS is 'QUERY DISK A'.

You can set all of your program function keys in your PROFILE EXEC, so they are set each time you load CMS. You can change a PF key setting any time during a terminal session, according to your needs.

```
CP SET PF5 IMMED XEDIT TEST FILE #BO# INPUT
```

For more details on setting PF keys, see the *VM/SP CP Command Reference for General Users* and the *VM/SP Terminal Reference*.

Display Screen Characteristics

Messages

During a CP or CMS session (other than an edit session) messages and warnings from the system operator or other users are highlighted. This distinguishes these messages from other output and lessens the possibility of important messages being lost or ignored.

A major feature of a 3270 display screen is the screen status area, which indicates, at all times that you are logged on, the current operating condition your virtual machine is in. Understanding the status conditions can help you use CMS on a 3270 more effectively.

Status Notices

The screen status area indicates one of the following conditions:

CP READ — After you log on, this is the first status message you see; it indicates that the terminal is waiting for a line to be read by the control program. You can enter only CP commands when the screen status area indicates a CP READ.

VM READ — This status indicates that your terminal is waiting for a line to be issued to your virtual machine; you may be in the CMS environment, in the edit or debug environments, or you may be executing a program or an EXEC that has issued a read to the console.

RUNNING

This status means that your virtual machine is operating. Once you have loaded CMS and are using the CMS environment, this status is almost continually in effect, even when you are not currently executing a command or program.

You can alter the way this works by using the AUTOREAD function of the SET command. When the AUTOREAD setting is OFF, (the default for display terminals), your terminal displays a RUNNING status after the execution of each CMS command. If you want the terminal to be in a VM READ status following each command, issue:

```
set autoread on
```

The ON setting is the default for typewriter terminals, because a read on a typewriter terminal must be accompanied by the unlocking of the keyboard.

The advantage of keeping your virtual machine in a running status even when it is not actually executing a program is that it makes your terminal ready to receive messages. If your terminal is waiting for a read, either from CP or from the virtual machine, and if a user or a program sends a message to your virtual console, then the message is not displayed until you use the ENTER key to enter a command or null line. When your machine is in a running status, the terminal console is always ready to accept messages.

If your virtual machine is in the CP environment, and you want your terminal to be in a running status, you can use the command:

```
cp sleep
```

To return to the CP READ status, you must press the PA1 key or the ENTER key.

MORE...

This status indicates that your display screen is full, but that there is more data to be displayed. This message, in addition to indicating that there is more data, gives you a chance to freeze your screen's current display so you can continue to examine it, if necessary.

When you see the screen is in a MORE... status, you can either:

- press the Clear, Cancel, or PA2 keys to clear the screen and see the next screen, or
- press the ENTER key to hold the screen in its present status. If you do not do either, then after 60 seconds, the screen is cleared and the next screen is displayed.

HOLDING

This indicates that you have pressed the ENTER key to freeze the screen. You must use the Cancel, Clear, or PA2 keys to erase this screen and go on to the next display.

A holding status also results if you have received a message that appeared on this screen. When the screen becomes full, it does not automatically pass to the next display after 60 seconds, but waits until you specifically clear the screen. (This feature ensures that any important messages you receive are not lost.)

NOT ACCEPTED — Indicates that you are trying to enter a command line but the terminal buffer is full and cannot accept it. This message is also issued when you attempt to use the 3270 COPY function and a printer is either not available or not ready.

Additional Display Screen Capabilities

The Extended Highlight feature and the Seven-Color feature are two added capabilities available for your use. Both features are available on the 3279 Models 2 and 3. If you are using 3278 Model 2, 3, 4, or 5, the options for both features will be accepted. However, only the highlight feature will be operable.

The CP SCREEN command (with its operands) allows you to chose one of three highlighting features (blinking, underscore, or reverse video) and one of seven different colors (red, green, blue, pink, turquoise, yellow, or white) for each screen area.

If you want the input area to be turquoise without highlighting, you should enter:

```
cp screen inarea turquois none
```

Or, if you want the input area pink and the status area yellow with the blinking highlight, you should enter:

```
cp screen inarea pink status yellow blink
```

The CP QUERY SCREEN command displays the color and extended highlight values currently in effect. For more details on the CP SCREEN command, see *VM/SP CP Command Reference for General Users* and for more details on the terminal display areas, see *VM/SP Terminal Reference*.

You can tailor your XEDIT screen colors with the XEDIT subcommand SET COLOR. Refer to the *System Product Editor Command and Macro Reference* for a description of the SET COLOR XEDIT subcommand.

How VM/SP Responds to Your Commands

CP and CMS respond differently to different types of requests. All CMS command responses (and all responses to CP commands that are entered from the CMS environment) are followed by the CMS ready message. The form of the ready message can vary, because it can be changed using the SET command. The long form of the ready message is:

```
R; T=7.36/19.89 09:26:11
```

If you have issued the command:

```
set rdymsg smsg
```

meaning, set the ready message to the short form, the ready message looks like:

```
R;
```

When you enter a command line incorrectly, you receive a message, describing the error. The ready message contains the last 5 digits (4 digits for a negative return code) from the command. For example:

```
R(00028);
```

indicates that the return code from the command was 28.

A ready message from the command may contain a negative return code; for example:

```
R(-0001);
```

indicates that the return code from the command was -0001.

Some Sample CP and CMS Command Responses

If you enter a CP or CMS command that requests information about your virtual machine, the response should be the information requested. For example, if you issue the command:

```
cp display g
```

CP responds by showing you the contents of your virtual machine's general registers, for example:

```
GPR  0 = 00000003 00003340 000007A0 00000003
GPR  4 = 00000848 C4404040 00000040 00002DF0
GPR  8 = 00000008 000132F8 00002BA0 00002230
GPR 12 = 00003238 FFFFFFFD 50013386 00000000
```

Similarly, if you issue the CMS command:

```
listfile * assemble c
```

you might receive the following information:

```
JUNK     ASSEMBLE C1
MYPROG   ASSEMBLE C1
```

If you enter a CP command to alter your virtual machine configuration or the status of your spool files, CP responds by telling you that the task is accomplished. The response to:

```
cp purge reader all
```

might be:

```
0004 FILES PURGED
```

Some CP commands, those that alter some of the characteristics of your virtual machine, give you no response at all. If you enter:

```
cp spool e class x hold
```

you receive no response from CP.

Certain CMS commands may issue prompting messages, to request you to enter more information. The SORT command, which sorts CMS disk files, is an example. If you enter:

```
sort in file a1 out file a1
```

you are prompted with the message:

```
DMSSRT604R ENTER SORT FIELDS:
```

and you can then specify which fields you wish the input records to be sorted on.

Getting Acquainted With CMS

If you have just logged on for the first time, and you want to try a few CMS commands, enter:

```
query disk a
```

the response might look like:

```
LABEL  CUU M STAT CYL TYPE BLKSIZE    BLKS USED-(%)  BLKS LEFT  BLKS TOTAL
PLC191 191 A  R/W   13 3380  1024           4864- 80       1181        6045
```

The response should tell you that you have an A-disk at virtual address 191; it also provides information such as how much room there is on the disk and how much of it is used. Again, if you receive an error message that indicates the disk may not be formatted, see "Formatting Virtual Disks."

Your A-disk is the disk you use most often in CMS, to contain your CMS files. Files are collections of data, and may have many purposes. You can invoke the System Product editor to create and modify files with the XEDIT command. To create a file named PARTY SUPPLIES, enter:

```
xedit party supplies
```

The display will look like Figure 1-1 on page 1-12.

```
PARTY    SUPPLIES A1  V 132  TRUNC=132 SIZE=0 LINE=0 COL=1 ALT=0
CREATING NEW FILE :




===== * * * TOP OF FILE * * *
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== * * * END OF FILE * * *




====> input_
                                                    X E D I T  1 FILE
```

Figure 1-1. Sample XEDIT Screen

On the command line (next to the arrow) type INPUT and press the ENTER key. The file is placed in input mode. The cursor is placed automatically on the first line in the input zone, where you can enter your data. You are writing input lines that are eventually going to be written onto your A-disk.

Enter the following data:

```
balloons
cake
party hats
ice cream
guests
```

```
 PARTY    SUPPLIES A1  V 132  TRUNC=132 SIZE=12 LINE=0 COL=1 ALT=0
INPUT MODE:




* * * TOP OF FILE * * *
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....
balloons
cake
party hats
ice cream
guests



====> * * * INPUT ZONE * * *
                                                    INPUT-MODE 1 FILE
```

Figure 1-2. Sample XEDIT Screen In INPUT Mode

Now, press the ENTER key, the screen moves up so that you can enter more data.

When you are finished entering data, press the ENTER key again to return to edit mode.

To keep this file in permanent storage, you type FILE on the command line and press the ENTER key. You should see a message that looks something like this:

```
R;
```

Even though the file has disappeared from your screen, the editor has saved it on your disk.

Let's check and see if the file was really saved. We'll use the LISTFILE command to list the files on your A-disk with the filename of PARTY. Enter:

```
listfile party
```

you should see the following:

```
PARTY SUPPLIES A1
```

Let's request a display of the file, using the TYPE command. Enter:

```
type party supplies
```

You should see the following:

```
balloons
cake
party hats
ice cream
guests
```

Since you really don't need this file, you can erase it from your permanent storage using the ERASE command. Enter:

```
erase party supplies
```

When you receive the ready message (R;), you know that the file was erased. Let's check to see if it really was erased. Use the LISTFILE command again to list the files on your A-disk with the filename of PARTY. Since you just erased the file, you'll receive the following message:

```
FILE NOT FOUND.
R(00028);
```

Most CMS commands create or reference disk files, and are as easy to use as the commands shown above. Your CMS disks are among the most important features in your VM/SP virtual machine.

Virtual Disks and How They Are Defined

Under VM/SP, a real direct access storage device (DASD) can be divided into many small areas, called minidisks. Minidisks, often called virtual disks, are defined in the VM/SP directory, as extents on real disks. For CMS applications, you never have to be concerned with the extents of your minidisks; when you use CMS-formatted minidisks, they are, for practical purposes, functionally the same as real disks. Minidisks can also be formatted for use with OS or DOS data sets or VSAM files.

You can have two types of disks, permanent and temporary.

Permanent disks — persist across logons and they are defined in the VM/SP directory entry for your virtual machine.

Temporary disks — are automatically destroyed at logoff. Temporary disks are those you define for your own virtual machine using the CP DEFINE command, or those attached to your virtual machine by the system operator.

Both permanent and temporary disks may be attached to your machine during a terminal session.

Permanent Virtual Disks

The VM/SP directory entry for your userid defines your permanent virtual disks. Each disk has associated with it an access mode specifying whether you can read and write on the disk or only read from it (its read/write status). Virtual disk entries in the VM/SP directory may look like the following:

```
MDISK  190  2314     000   050  CMS190  R
MDISK  191  3330     010   005  BDISKE  W
MDISK  194  3330     010   020  CMS001  W
MDISK  195  FB-512   1000  500  FBDISK  W
MDISK  198  3330     050   010  CMS192  W
MDISK  19E  3330     010   050  CMS19E  R
```

The first two fields describe the device MDISK (minidisk) and the virtual address of the device. Virtual addresses (shown above as 190, 191, and so on), are the names by which you and VM/SP identify the disk. Each device in your virtual machine has an address which may or may not correspond to the real address of the device on the VM/SP system.

The third field specifies the device type of your virtual disk. For count-key-data devices, the fourth and fifth fields specify the starting real cylinder at which your virtual disk logically begins and the number of cylinders allocated to your virtual disk, respectively. For FB-512 devices, the fourth field specifies the starting real block numbers where your virtual disk begins, and the fifth field is the number of blocks allocated to your virtual disk.

The sixth field is the label of the real disk on which the virtual disk is defined and the seventh field is a letter specifying the read/write mode of the disk; "R" indicates that the disk is a read-only disk, and "W" indicates that you have read/write privileges. The MDISK control statement of the Directory Service Program is described in the *VM/SP Planning Guide and Reference.*

Defining Temporary Virtual Disks

Using the CP DEFINE command, you can attach a temporary disk to your virtual machine for the duration of a terminal session. The following command allocates a 10-cylinder temporary disk from a 3330 device and assigns it a virtual address of 291:

```
cp define t3330 as 291 cyl 10
```

When you define a minidisk, you can choose any valid address that is not already assigned to a device in your virtual machine. Valid addresses for minidisks range from 001 through 5FF, for a virtual machine in basic control mode.

Formatting Virtual Disks

Before you can use any new virtual disk, you must format it. This applies to new disks that have been assigned to you and to temporary disks that you have allocated with the CP DEFINE command. When you issue the FORMAT command you must use the virtual address you have defined for the disk and assign a CMS mode letter, for example:

```
format 291 c
```

CMS then prompts you with the following message:

```
DMSFOR603R    FORMAT WILL ERASE ALL FILES ON DISK 'C(291)'.
DO YOU WISH TO CONTINUE? (YES|NO):
```

You respond:

```
yes
```

CMS then asks you to assign a label for the disk, which may be anything you choose. Labels can have a maximum of 6 characters. When the message:

```
DMSFOR605R ENTER DISK LABEL:
```

is issued, you respond by supplying a disk label. For example, if this is a temporary disk, you might enter:

```
scrtch
```

CMS then erases all the files on that disk, if any existed, and formats the disk for your use and displays the following messages:

```
FORMATTING DISK 'C'
'10' CYLINDERS FORMATTED ON 'C(291)'.
R; T=0.15/1.60  11:26:03
```

The FORMAT command should only be used to format CMS disks, that is, disks you are going to use to contain CMS files. In addition, this command allows you a choice of physical disk block size as an option. Refer to the *VM/SP CMS Command and Macro Reference* for details. Format disks for OS, DOS, or VSAM applications, using the Device Support Facilities. See the *VM/SP Operator's Guide* for details.

Sharing Virtual Disks: Linking

Since only one user can own a virtual disk, and there are many occasions that require users to share data or programs, VM/SP allows you to share virtual disks, on either a permanent or temporary basis, by "linking."

Permanent links can be established for you in your VM/SP directory entry. These disks are then a part of your virtual machine configuration every time you log on. You can also have another user's disk temporarily added to your configuration by using the CP LINK command. For example, if you have a program that uses data that resides on a disk identified in userid DATA's configuration as a 194, and you know that the password assigned to this disk is GO, you could issue the command:

```
cp link to data 194 as 198 r pass= go[1]
```

DATA's 194 disk is then added to your virtual machine configuration at virtual address 198.

The "R" in the command line indicates the access mode; in this case, it tells CP that you only want to read files from this disk and you will not be allowed to write on it. If you try to issue this command when someone already has write access to that disk, you will not be able to establish the link. If you want to link to DATA in any event, you can reissue the LINK command using the access mode RR:

```
cp link data 194 198 rr go[1]
```

The keywords "TO," "AS," and "PASS=" are optional; you do not have to specify them.

However, note that using the RR access allows one user to read a disk while another is updating the same disk at the same time. This may produce unpredictable results.

You can also use the CP LINK command to link to your own disks. For example, if you log on and discover that another user has access to one of your disks, you may be given read-only access, even if it is a read/write disk. You can request the other user to detach your disk from his virtual machine, and after he has done so, you can establish the link:

```
cp link * 191 191
```

When you link to your own disks, you can specify the userid as * and you do not need to specify the access mode or a password.

---

1 The password cannot be entered on the command line if the password suppression facility was specified when your system was installed.

You can find more information about the CP LINK command and CP access modes in *VM/SP CP Command Reference for General Users.*

Identifying Your Disk To CMS: Accessing

LINK and DEFINE are CP commands: they tell CP to add DASD devices to your virtual machine configuration. CMS must also know about these disks, and you must use the ACCESS command to establish a filemode letter for them:

```
access 194 b
```

CMS uses filemode letters to manage your files during a terminal session. By using the ACCESS command you can control:

- Whether you can write on a disk or only read from it (its read/write status).
- The library search order for programs executing in your virtual machine.
- Which disks are to contain the new files that you create.

If you want to know which disks you currently have access to, issue the command:

```
query search
```

You might see the following display:

```
PLC191   191  A     R/W
DAT194   198  B     R/O
CMS190   190  S     R/O
CMS19E   19E  Y     R/O
```

The first column indicates the label on the disk (assigned when the disk is formatted), and the second column shows the virtual address assigned to it.

The third column contains the filemode letter. All letters of the alphabet are valid filemode letters.

The fourth column indicates the read/write status of the disk. The 190 and 19E disks in this example are read-only disks that contain the CMS nucleus and disk-resident commands for the CMS system. You will probably use your 191 (A) disk as your primary read/write work disk.

Releasing Virtual Disks

When you no longer need a disk that you linked or temporarily accessed, then release that disk. To release a disk, use the CMS RELEASE command:

```
release c
```

When you want to assign a currently active filemode letter to another disk, issue the ACCESS command to assign that filemode letter to another disk. It is not necessary to release an accessed disk prior to accessing another disk with the same filemode.

When you no longer need disks in your virtual machine configuration, use the CP command DETACH to disconnect them from your virtual machine:

```
cp detach 194
cp detach 291
```

If you are going to release and detach the disk at the same time, you can use the DET option of the RELEASE command:

```
release 194 (det
```

When you logoff the disks are released automatically. For more information on controlling disks in CMS, see "Chapter 3. The CMS File System."

Console Output

When you use a 3270 terminal as your virtual machine console, you do not ordinarily retain a console log, as you do on typewriter terminal. There may be many circumstances in which you need a printed record of your console output, whether it be to obtain a copy of program-generated output, or to retain a record of CP and/or CMS commands that resulted in an error condition. There are two techniques you can use in VM/SP to obtain hardcopy representations of display terminal sessions: spooling console output and the 3270 copy function.

Spooling Console Output

The CP SPOOL command provides the CONSOLE operand, which allows you to begin and end console spooling. You enter:

```
cp spool console start
```

when you want to begin recording your terminal session, and:

```
cp spool console stop
```

when you have finished. In between, you can periodically close the console file to release for printing whatever has been spooled thus far:

```
cp spool console close
```

Other operands that you can enter are the same as you might specify for any printer file, such as CLASS, COPY, CONT, and HOLD.

An alternate technique is to spool your console to your own virtual reader:

```
cp spool console start * class a
```

Then, when you close the console file, instead of being released to the CP printer spool file queue, it is routed to your virtual reader, and you can load it onto your A-disk as a CMS disk file:

```
readcard console file
```

You can then use the editor to examine it (or to delete sections you don't need) and use the PRINT command to spool it to the printer.

3270 COPY Function

If you are using a 3270 display terminal, and you have available a 3284, 3286, 3287, 3288, or 3289 printer, you can copy the full screen display currently appearing on the screen. To copy the screen, you have to assign the copying function to a program function key, with the SET command:

```
cp set pf9 copy
```

Note: The PF key copy function is not available if the printers are dedicated.

Then, whenever you want to copy a screen display, you can press the PF9 key (or whichever key you set). The display is printed on any 3270 display printer that is attached to the same remote control unit as the display terminal. If, when you press the PF key, the screen status area indicates NOT ACCEPTED, it means that the printer is either not ready or not available. When you press the PF key and receive no response, it means that the screen has been copied.

There is a print matrix available to the 3274 and 3276 user that allows control of the display to printer operations. In addition, a local print key is provided on the 3274 that can be used for copy operations.

Figure 1-3 an example of a 3270 screen display that could be copied on the printer. When you use the copy function to copy a screen, all 24 lines of the display screen are copied; the screen status area (indicated as RUNNING in Figure 1-3) is blank if the 3270 is locally attached. If the 3270 is remotely attached, the entire screen including the screen status area, is copied. You can use the user input area of your screen to key in comments, or your name or userid, if several users are spooling copy files.

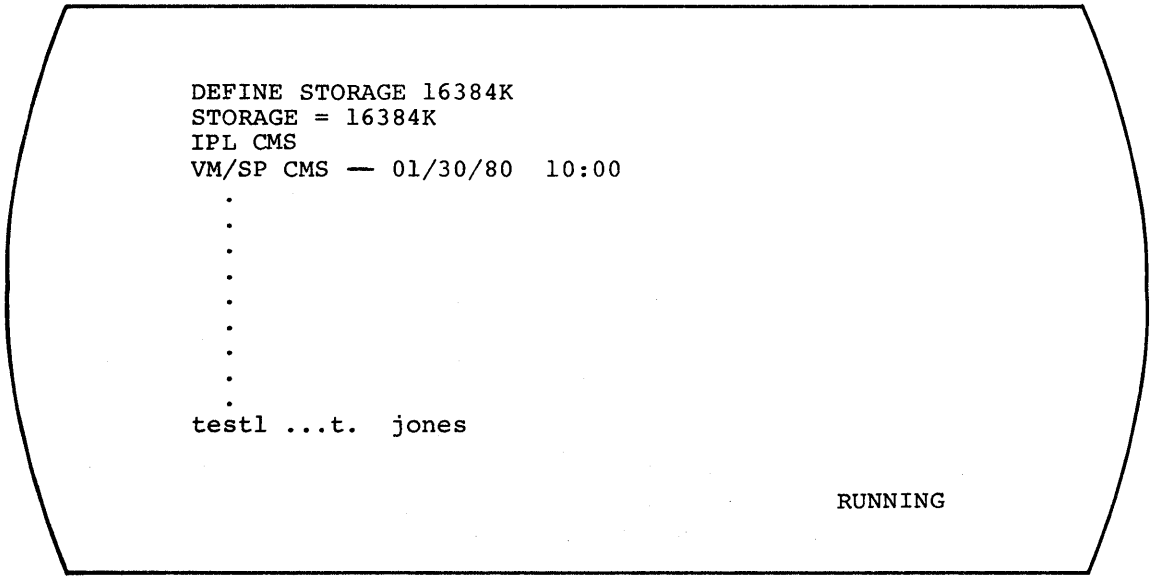


Figure 1-3. 3270 Screen Display

Chapter 2. VM/SP Environments and Mode Switching

When you are using VM/SP, your virtual machine can be in one of two possible "environments," the control program (CP) environment or the virtual machine environment, which may be CMS. The CMS environment has several subenvironments, sometimes called "modes." Each environment or subenvironment accepts particular commands or subcommands, and each environment has its own entry and exit paths, responses and error messages. If you have a good understanding of how the VM/SP environments are related, you can learn to change environments quickly and use your virtual machine efficiently.

This section introduces the CP and CMS environments that you use and describes:

- Entry and exit paths
- Command subsets that are valid as input

Figure 2-1 summarizes the VM/SP command environments and lists the commands and terminal paths that allow you to go from one environment to another.

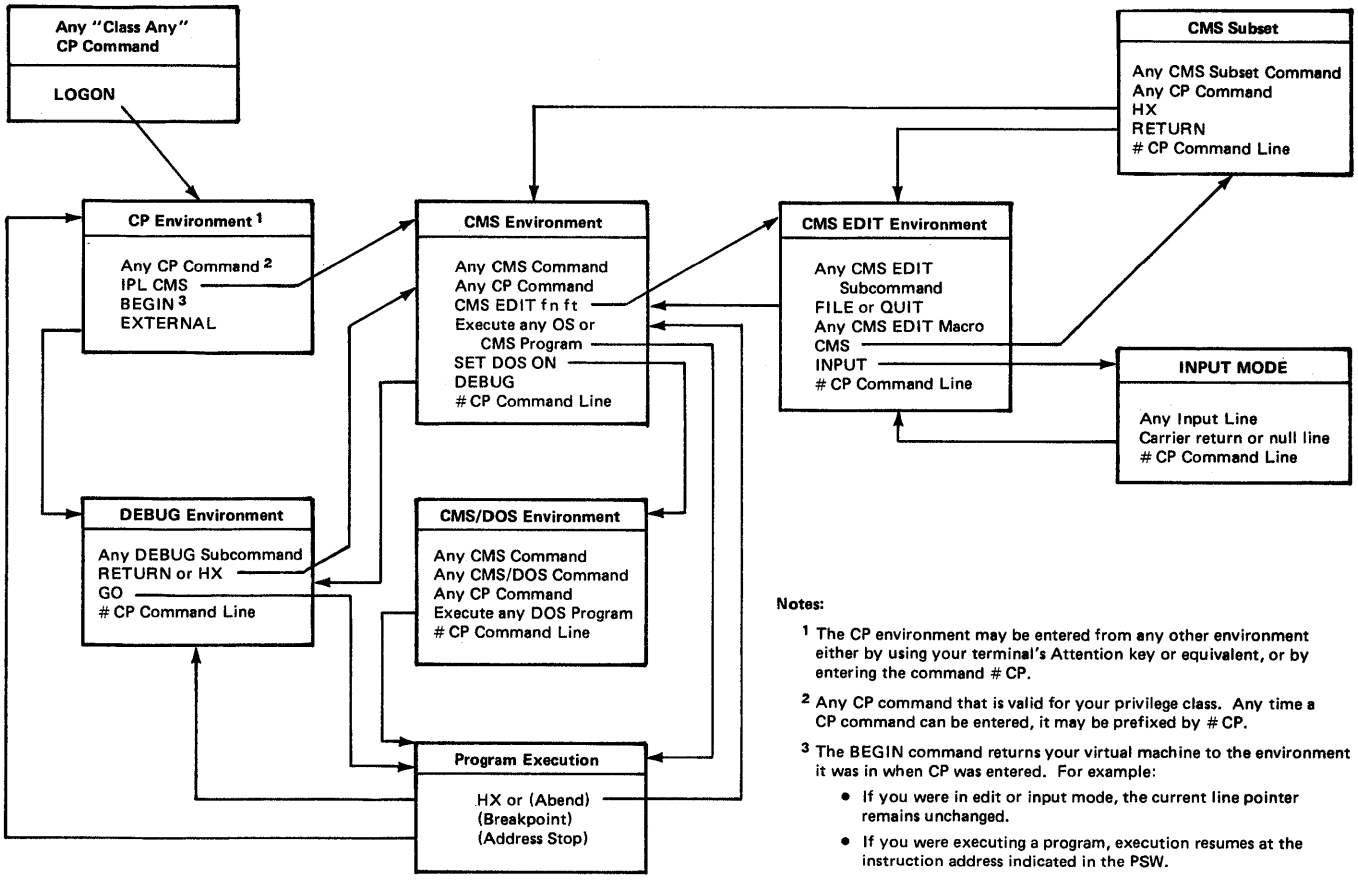


Figure 2-1. VM/SP Environments and Mode Switching

With the exception of input mode in the edit environment, you can always determine which environment your virtual machine is in by pressing the Return or Enter key on a null line. The responses you receive and the environments they indicate, are:

Response	Environment
CP	CP
CMS	CMS
CMS (DOS ON)	CMS/DOS
CMS SUBSET	CMS Subset
DEBUG	Debug

The CP Environment

When you log on to VM/SP, your virtual machine is in the CP environment. In this environment, you can enter any CP command that is valid for your privilege class. This publication assumes that you are a general, or class G, user. You can find information about the commands that you can use in the *VM/SP CP Command Reference for General Users.*

Only CP commands are valid terminal input in the CP environment. You can, however, preface a CP command line with the characters "CP" or "#CP," followed by one or more blanks, although it is not necessary. These functions are described under "The CMS Environment."

You can enter CP commands from other VM/SP environments. There may be times during your terminal session when you want to enter the CP environment to issue one or more CP commands. You can do this from any other environment by doing either of two things:

1. Issue the command:

   ```
   #cp
   ```

2. Use your terminal's Attention key (PA 1 or equivalent). On a 2741 terminal, you must normally press the Attention key twice, quickly, to enter the CP environment.

The following message indicates that your virtual machine is in the CP environment:

```
CP
```

After entering whatever CP commands you need to use, you return your virtual machine to the environment or mode that it came from by using the CP command:

```
cp begin
```

which, literally, begins execution of your virtual machine.

The CMS Environment

You enter the CMS environment from CP by issuing the IPL command, which loads CMS into your virtual storage area. If you are planning to use CMS for your entire terminal session, you should not have to IPL again unless a program failure forces you into the CP environment.

When you issue the IPL command, specify the named system CMS at your installation. For example:

```
cp ipl cms
```

When your virtual machine is in the CMS environment, you can issue any CMS command and any of the CP commands that are valid for your user privilege class. You can also execute many of your own OS or DOS programs; the ways you can execute programs are discussed in Chapter 9, "Developing OS programs under CMS" and Chapter 10, "Developing VSE Programs Under CMS."

You can enter CP commands from CMS in any of the following ways:

- Using the implied CP function of CMS (See *Note.*)
- With the CP command
- With the #CP function

Note: For the most part, you may enter any CP command directly from the CMS environment. This implied CP function is controlled by an operand of the CMS SET command, IMPCP. You can determine whether the implied CP function is in effect for your virtual machine by entering the command:

```
query impcp
```

If the response is:

```
IMPCP      = OFF
```

you can change it by entering:

```
set impcp on
```

When the implied CP function is set off, you must use either the CP command or the #CP function to enter CP commands from the CMS environment. CP commands that you execute in EXEC procedures must always be prefaced by the CP command, regardless of the implied CP setting. An example of using the CP command is:

```
cp close punch
```

When you issue CP commands from the CMS environment either implicitly or with the CP command, you receive, in addition to the CP response (if any), the CMS ready message. If you use the #CP function, discussed next, you do not receive the ready message.

You can preface any CP command line with the characters "#CP," followed by one or more blanks. When you enter a CP command this way, the command is processed by CP immediately; it is as if your virtual machine were actually in the CP environment.

EDIT and CMS Subset

The System Product editor and the CMS editor are VM/SP facilities that allow you to create and modify data files that reside on CMS disks. The editor environment, more commonly called the edit environment, is entered when you issue either of the CMS commands, XEDIT or EDIT, specifying the identification of a data file you want to create or modify. Complete information about the System Product editor,

invoked via the XEDIT command, is found in the *VM/SP System Product Editor Command and Macro Reference* and the *VM/SP System Product Editor User's Guide*. For introductory tutorial information about editing, refer to the *VM/SP CMS Primer*.

```
xedit party supplies
```

is an example of how you would enter the edit environment to either create a file called PARTY SUPPLIES or to make changes to a disk file that already exists under that name.

When you enter the edit environment your virtual machine is automatically in edit mode, where you can only issue XEDIT subcommands, CMS commands, or CP commands prefaced by "#CP." After you enter the XEDIT subcommand:

```
input
```

data lines that you enter are considered input to the file. To return to edit mode, you must enter a null line.

If you issue the XEDIT subcommand:

```
cms
```

the editor responds:

```
CMS SUBSET
```

and your virtual machine is in CMS subset mode. When in CMS subset mode, you can issue any valid CMS subset command, that is, a CMS command that is allowed in CMS subset mode. These include:

```
ACCESS          NAMEFIND        RDR
CP              NAMES           RDRLIST
DISK            NOTE            READCARD
ERASE           NUCXDROP        RECEIVE
EXEC            NUCXLOAD        SENDFILE
EXECIO          NUCXMAP         SET
FILELIST        PEEK            STATE
GLOBALV         PRINT           STATEW
IDENTIFY        PUNCH           TELL
LISTFILE        QUERY           TYPE
```

You can also issue CP commands. To return to edit mode, you use the special CMS subset command, RETURN. If you enter the Immediate command HX, your editing session terminates abnormally and your virtual machine returns to the CMS environment.

The following is a description of the CMS subset environment as entered by the 'CMS' EDIT subcommand.

The CMS Subset Environment: When entering CMS subset mode either for a single command or until the string 'RETURN' is entered, the following processing is done to ensure that the previous environment is preserved. Upon entry to subset, a check is made to determine if this entry would constitute a recursion, if so, return code 1 is returned.

1. STAE, SPIE, and STAX information is saved and then cleared.

2. The OS environment settings are saved and then cleared so that any module that issues an OSRESET based on these flags will not do so.

3. The read and write pointers from any currently opened files are saved.

4. All files are then closed by a 'FINIS * * *', but files with a filemode of 3 are not erased.

5. Any FSTs that were built by a previous call to STATE are saved.

If the entry to subset was just for the execution of a single command, the entry message is suppressed and the next command is executed immediately. But, of the request was to enter CMS subset for an indefinite duration, an announcement of entry to the CMS subset environment is made. This is done so that a strict differentiation from the strict command environment is given.

The principle difference in subset is the restriction that any command executed may not use any storage other than DMSFREE storage and the transient area. This protects programs which may be running in the USER AREA. Also, any ready message issued from subset is in the abbreviated form (i.e. identical to SET RDYMSG SMSG) so that program timing information is not effected for the command currently in progress at the time of subset entry.

Upon termination of CMS subset mode any settings or values that were saved upon entry to subset are restored.

When you are finished with an edit session, you return to the CMS environment by issuing the FILE subcommand, which indicates that all modifications or data insertions that you have made should be written onto a CMS disk. Otherwise, you can issue the subcommand QUIT, which tells the editor not to save any modifications or insertions made since the last time the file was written.

More detailed information about EDIT subcommands and how to use the CMS editor is contained in this publication in "Appendix A" and in the *VM/SP CMS Command and Macro Reference*.

DEBUG

CMS DEBUG is a special CMS facility that provides subcommands to help you debug programs at your terminal. Your virtual machine enters the debug environment when you issue the CMS command:

```
debug
```

You may want to enter this command after you have loaded a program into storage and before you begin executing it. At this time you can set "breakpoints," or address stops, where you wish to halt your program's execution so that you can examine and change the contents of general registers and storage areas. When these breakpoints are encountered, your virtual machine is placed in the debug environment. You can also enter the debug environment by issuing the CP EXTERNAL command, which causes an external interrupt to your virtual machine.

Valid DEBUG subcommands that you can enter in this environment are:

```
BREAK          GO             RETURN
CAW            GPR            SET
CSW            HX             STORE
DEFINE         ORIGIN         X
DUMP           PSW
```

You can also use the #CP function in the debug environment to enter CP commands.

You leave the debug environment in any of the following ways:

- If the program you are running completes execution, you are returned to the CMS environment.
- If your virtual machine entered the debug environment after a breakpoint was encountered, it returns to CMS when you issue the DEBUG subcommand:

  ```
  hx
  ```

  To continue the execution of your program, you use the DEBUG subcommand:

  ```
  go
  ```

- If your virtual machine is in the debug environment and is not executing a program, the DEBUG subcommand:

  ```
  hx
  ```

  returns it to the CMS environment.

Refer to Chapter 13, "Debugging Your Program Using VM/SP" for more information.

CMS/DOS

If you are a VSE user, the CMS/DOS environment provides you with all the CMS interactive functions and facilities, as well as special CMS/DOS commands which simulate VSE functions. The CMS/DOS environment becomes active when you issue the command:

```
set dos on
```

When your virtual machine is in the CMS/DOS environment you can issue any command line that would be valid in the CMS environment, including the facilities of XEDIT, DEBUG, and EXEC, except for CMS commands or program modules that load and/or execute programs that use OS macros or functions.

The following commands are provided in CMS/DOS to test and develop DOS programs, and to provide access to VSE libraries:

```
ASSGN          DOSPLI         LISTIO
CATCHECK       DSERV          OPTION
DLBL           ESERV          PSERV
DOSLIB         FETCH          RSERV
DOSLKED        FCOBOL         SSERV
```

Your virtual machine leaves the CMS/DOS environment when you issue the command:

```
set dos off
```

If you reload CMS (with an IPL command) during a terminal session, you must also reissue the SET DOS ON command. For more information about the CMS/DOS environment, see Chapter 10, "Developing VSE Programs Under CMS."

Interrupting Program Execution

When you are executing a program under CMS or executing a CMS command, your virtual machine is not available for you to enter commands. There are, however, ways in which you can interrupt a program and halt its execution, either temporarily, in which case you can resume its execution, or permanently, in which case your virtual machine returns to the CMS environment. In both cases, you interrupt execution by creating an "attention interruption," which may take two forms:

- An attention interruption to your virtual machine operating system
- An attention interruption to the control program

These situations result in what are known as virtual machine (VM) or control program (CP) "reads" being presented to your virtual console. The two keys on your 3270 keyboard that signal interruptions are the PA1 key -- REQ key on a 3278 Model 2A -- and the Enter key. Throughout this publication, interruption signaling has been described in terms of the Attention key.

On a typewriter terminal, the Attention key, pressed once, causes a virtual machine interruption (if the terminal mode is set to VM); you must use it when you want to enter an Immediate command, such as HT or HX. On a display terminal, you can enter these commands whenever your virtual machine is in a running status, without having to signal an interruption before you enter the command.

Sometimes, however, if your terminal is displaying output very rapidly, you must wait until the screen is full and the screen status area indicates a MORE... status before you attempt to enter the HT or HX command.

The Enter key can also be used as an interruption signaling key. If you press it once when your virtual machine is running, you will place your virtual machine in the VM READ status, so you can enter a command line.

You can enter the CP environment by pressing the PA1 key. Whenever you press this key, your virtual machine is placed in a CP READ status, and you can enter any CP command. From the CP environment, you must use the CP command BEGIN to resume execution of your virtual machine. On a typewriter terminal, the keyboard unlocks when a read occurs.

Whether you have to press the Attention key once or twice depends on the terminal mode setting in effect for your virtual machine. This setting is controlled by the CP TERMINAL command:

```
cp terminal mode vm
```

The setting VM is the default for virtual machines; you do not need to specify it. The VM setting indicates that one depression of the Attention key sends an interruption to your virtual machine, and that two depressions results in an interruption to the control program (CP).

The CP setting for terminal mode, which is the default for the system operator, indicates that one depression of the Attention key results in an interruption to the control program (CP). If you are using your virtual machine to run an operating system other than CMS, you might wish to use this setting. Issue the command:

```
cp terminal mode cp
```

Virtual Machine Interruptions

While a command or program is executing, if you press the ENTER key on a 3270 (or the Attention key once on a 2741), you have created a virtual machine interruption.

Halting Screen Displays

When your terminal is displaying successive screens of output from a program or a CMS command, you can use the HT or HX Immediate commands to halt the display or the execution of the command, respectively. If your terminal is writing the information at a very rapid rate, you may have difficulty entering the HT or HX command. In these circumstances, you can use the PA1 key -- REQ key on a 3278 Model 2A -- or press the Enter key twice to force your terminal to a CP READ status. Then, you can use the CP command ATTN or REQUEST to signal a virtual machine read. When the screen status area indicates VM READ, you can enter HX or HT. The program halts execution, your terminal will accept an input line, and you may:

- Terminate the execution of the program by issuing an Immediate command to halt execution:

  ```
  hx
  ```

  The HX command causes the program to abnormally terminate (abend).

- Enter a CMS command. The command is stacked in a console stack and is processed by CMS when your program is finished executing and the next virtual machine read occurs. For example:

  ```
  print abc listing
  ```

  After you enter this line, the program resumes execution.

- If the program is directing output to your terminal and you wish only to halt the terminal display, use the Immediate command:

  ```
  ht
  ```

  The program resumes execution. Terminal output can also be suppressed immediately when you enter a command by placing #HT at the end of the command line. The logical line end character (#) allows the Immediate command HT to be accepted; program execution proceeds without typing.

  You can, if you want, cause another interruption and request that typing be resumed by entering the RT (resume typing) command:

  ```
  rt
  ```

- Enter a null line; your program continues execution. The null line is stacked in the console stack and read by CMS as a stacked command line.

HX, HT, and RT are three of the CMS Immediate commands. They are "immediate" because they are executed as soon as they are entered. Unlike other commands, they are not stacked in the console stack. You can only enter an Immediate command following an attention interruption.

Control Program Interruptions

You can interrupt a program and enter the CP environment directly by pressing the PA1 key on a 3270 or by pressing the Attention key twice quickly, on a 2741. Then, you can enter any CP command. To resume the program's execution, issue the CP command:

```
cp begin
```

If your terminal is operating with the terminal mode set to CP, pressing the Attention key once places your virtual machine in the CP environment.

Address Stops and Breakpoints

A program may also be interrupted by an instruction address stop, which you specifically set by the CP command ADSTOP. For example, if you issue the command:

```
cp adstop 201ea
```

an address stop is set at virtual storage location X'201EA'. When your program reaches this address during its execution, it is interrupted and your virtual machine is placed in the CP environment, where you can issue any CP command, including another ADSTOP command, before resuming your program's execution with the CP command BEGIN.

Breakpoints, similar to address stops, are set using the DEBUG subcommand BREAK, which you issue in the debug environment before executing a program. For example, if you issue:

```
break 1 201ae
```

your program's execution is interrupted at this address and your virtual machine is placed in the debug environment. You can then enter any DEBUG subcommand. To resume program execution, use the DEBUG subcommand GO. If you want to halt execution of the program entirely, use the DEBUG subcommand HX, which returns your virtual machine to the CMS environment. You can find more information about setting address stops and breakpoints in Chapter 13, "Debugging Your Program Using VM/SP."

Using APL

If you have a 3277 or 3278 display station equipped with an APL keyboard, you can use APL on a 3270 terminal in CMS. You invoke the APL virtual machine by issuing the command specified in the VS APL Program Product documentation. This command invokes the VS APL-CMS interface program. You are then prompted to press the APL On/Off key which is on your terminal; pressing this key changes the keyboard to APL character input mode. You are then prompted to press the Enter key to continue.

EBCDIC or APL characters can always be displayed; the APL On/Off key does not change this. The VS APL-CMS interface program issues the TERMINAL APL ON command for you and selects the appropriate translation tables. The

TERMINAL APL ON command automatically forces a TERMINAL TEXT OFF condition. The interface program then invokes the VSAPL program. When the VSAPL ready message appears on the screen, you can use APL.

You can send a copy of your display screen to a locally or remotely attached printer. Be sure that the printer you send your output to has the APL feature installed; if it does not, the APL characters are not printed. Most system printers do not have an APL print chain; therefore you may need to use the copy function to direct your screen output displays to a 3284, 3286, or 3287 printer.

Error Situations

If you do not have the APL hardware feature installed on your 3277 or 3278 but you invoke APL:

- The VSAPL program is invoked and the TERMINAL APL ON command is issued.
- You cannot communicate with the VSAPL program.
- Any APL characters that are written to the screen appear as blanks.

If you have the APL feature installed on your terminal, but invoke APL manually without issuing the TERMINAL APL ON command or issue TERMINAL APL OFF at sometime during APL processing:

- The VSAPL program is activated.
- You cannot communicate with the VSAPL program.
- Any APL characters written to the screen appear as blanks.

If you attempt to use the APL O/S (overstrike) key when the APL hardware key is set off, it acts as a backtab key and repositions the cursor to the beginning of the user input area.

Leaving the APL Environment

Issue the APL command:

```
) OFF
```

to log off VM/SP.

Issue the APL command:

```
) OFF HOLD
```

to return to CMS. This APL command invokes the VS APL-CMS interface program, which:

- Issues the TERMINAL APL OFF command
- Prompts you to press the APL hardware key
- Returns to CMS

Note: The APL hardware feature is a key, not a switch. Each time you press the APL key you reverse its on/off setting. To determine whether

APL is on or off, press a key that represents a special APL character. If the character displayed is an APL character, the hardware APL feature is set on. If the character displayed is a non-APL character, you must press the APL key once to set the APL feature on.

Using the 3277 Text Feature

If you have a 3277 or 3278 display station equipped with the Data Analysis Text keyboard, you can key in, as well as display, all of the special text characters. For a description of these characters, see the *VM/SP Terminal Reference*. These characters are in addition to those available with standard EBCDIC 3270 terminals. If you have a suitably equipped printer attached to your 3270, you can use the SET PFnn COPY function to obtain a printed copy of the screen.

When you want to activate the text feature, and use the special characters, enter the command:

```
cp terminal text on
```

The TERMINAL TEXT ON command automatically forces the TERMINAL APL OFF command. Now, you can use any of the special characters when you enter, change, or locate text lines in a file.

You leave the special text environment by entering the command:

```
cp terminal text off
```

Error Situations

If you do not have the appropriate text hardware feature on your 3270, but attempt to display a file that contains the characters, the characters appear as blanks on a 3277, and as hyphens on a 3276 and a 3278.

If you inadvertently issue the TERMINAL TEXT ON command while using a terminal that does not have the text capability, you must do the following to return to normal operating procedures:

1. Press the PA1 key to enter the CP environment.
2. Key in, in uppercase letters only, the command line:

   ```
   TERMINAL TEXT OFF
   ```

Notes:

1. The 3270 text hardware feature is activated by a key, not a switch. Each time you press the TEXT On/Off key, you reverse its setting. When the red light on the text keyboard is illuminated, the text feature is on.

2. Compound characters, such as a .hw character/-backspace/-character combination, are still entered and displayed as three characters. The screen position occupied by the backspace character appears as a blank because the character (X'16') is nondisplayable.

Chapter 3. The CMS File System

The file is the essential unit of data in the CMS system. CMS disk files are unique to the CMS system and cannot be read or written using other operating systems. When you create a file in CMS, you name it using a file identifier. The file identifier consists of three fields:

- `filename (fn)`
- `filetype (ft)`
- `filemode (fm)`

When you use CMS commands and programs to modify, update, or reference files, you must identify the file by using these fields. Some CMS commands require you to enter only the filename, or the filename and filetype; others require you to enter the filemode field as well. This section contains information about the things you must consider when you give your CMS files their identifiers, notes on the file system commands that create and modify CMS files, and additional notes on using CMS disks.

CMS File Formats

The CMS file management routines write CMS files on disk in fixed physical blocks, regardless of whether they have fixed- or variable-length records. For most of your CMS applications, you never need to specify either a logical record length and record format or block size when you create a CMS file.

When you create a file using one of the CMS editors, the file has certain default characteristics, based on its filetype. The special filetypes recognized by the editor, and their applications, are discussed under "What are Reserved Filetypes"?

VSAM files written by CMS are in the same format as VSAM files written by OS/VS or VSE and are recognized by those operating systems. You cannot, however, use any CMS file system commands to read and write VSAM files, because VSAM file formats are unique to the virtual storage access method.

How CMS Files Get Their Names

When you create a CMS file, you can give it any filename and filetype you wish. The rules for forming filenames and filetypes are:

- The filename and filetype can each be from one to eight characters.
- The valid characters are A-Z, a-z, 0-9, $, #, @, +, - (hyphen), : (colon), and _ (underscore).

When you enter a command line into the VM/SP system, VM/SP translates your input line by either the user defined input table or by the uppercase table. See the CMS SET INPUT command in the *VM/SP CMS Command and Macro Reference.* If you do not have an input table, you can just enter the command line in lowercase and VM/SP translates your input line into uppercase characters.

Note: When defining input characters be sure that you will not end up with a fileid containing invalid characters.

The # and @ characters are line editing symbols in VM/SP; when you use them to identify a file, you must precede them with the logical escape symbol ("). See Appendix C, "Considerations for Line Mode Terminals" for a list of logical line editing symbols.

The third field in the file identifier, the filemode, indicates the mode letter (A-Z) currently assigned to the virtual disk on which you want the file to reside. When you use the editor to create a file, and you do not specify this field, the file you create is written on your A-disk, and has a filemode letter of A.

The filemode letter, for any file, can change during a terminal session. For example, when you log on, your virtual disk at address 191 is accessed as your A-disk, so a file on that disk named SPECIAL EVENTS has a file identifier of:

```
SPECIAL EVENTS A
```

If, however, you later access another disk as your A-disk, and access your 191 as your B-disk, then this file has a file identifier of:

```
SPECIAL EVENTS B
```

Duplicate Filenames or Filetypes

You can give the same filename to as many files on a given disk as you want, as long as you assign them different filetypes. Or you can create many files with the same filetype but different filenames.

For the most part, filenames that you choose for your files have no special significance to CMS. If, however, you choose a name that is the same as the name of a CMS command, and the file that you assign this name to is an executable module or EXEC procedure, then you may encounter difficulty if you try to execute the CMS command whose name you duplicated.

For an explanation of how CMS identifies a command name, see "CMS Command Search Order" later in this chapter.

Many CMS commands allow you to specify one or more of the fields in a file identifier as an asterisk (*) or equal sign (=), which identify files with similar fileids.

Using Asterisks (*) in Fileids

Some CMS commands that manipulate disk files allow you to enter the filename and/or filetype fields as an asterisk (*), indicating that all files of the specified filename/filetype are to be modified. These commands are:

```
COPYFILE    RENAME
ERASE       TAPE DUMP
```

For example, if you specify:

```
erase * test a
```

all files with a filetype of TEST on your A-disk are erased. The LISTFILE command allows you to request similar lists. If you specify an asterisk for a filename or filetype, all of the files of that filename or filetype are listed. There is

```
listfile t* assemble
```

produces a list of all files on your A-disk with filenames beginning with the letter T and with the filetype of assemble. The command:

```
listfile tr* a*
```

produces a list of all files on your A-disk with filenames beginning with the letters TR and with filetypes beginning with the letter A.

Equal Signs in Output Fileids

The COMPARE, COPYFILE, RENAME, and SORT commands allow you to enter output file identifiers as equal signs (=), to indicate that it is the same as the corresponding input file identifier. For example:

```
copyfile myprog assemble b = = a
```

copies the file MYPROG ASSEMBLE from your B-disk to your A-disk, and uses the same filename and filetype as specified in the input fileid for those positions in the output fileid.

Similarly, if you enter the command:

```
rename temp * b perm = =
```

all files with a filename of TEMP are renamed to have filenames of PERM; the existing filetypes of the files remain unchanged.

What Are Reserved Filetypes?

For the purposes of most CMS commands, the filetype field is used merely as an identifier. Some filetypes, though, have special uses in CMS; these are known as "reserved filetypes."

Nothing prevents you from assigning any of the reserved filetypes to files that are not being used for the specific CMS function normally associated with that filetype.

Some reserved filetypes also have special significance to the System Product editor and the CMS editor. When you use either the XEDIT or the EDIT command to create a file with a reserved filetype, the editor assumes various default characteristics for the file, such as record length and format, tab settings, translation to uppercase, truncation column, and so on.

Filetypes for CMS Commands

Reserved filetypes sometimes indicate how the file is used in the CMS system: the filetype ASSEMBLE, for example, indicates that the file is to be used as input to the assembler; the filetype TEXT indicates that the file is in relocatable object form, and so on. Many CMS commands assume input files of particular filetypes, and require you to enter only the filename on the command line. For example, if you enter:

```
synonym test
```

CMS searches for a file with a filetype of SYNONYM and a filename of TEST. A file named TEST that has any other filetype is ignored.

Some CMS commands create files of particular filetypes, using the filename you enter on the command line. The language processors do this as well; if you are recompiling a source file, but wish to save previous output files, you should rename them before executing the command.

Figure 3-1 lists the filetypes used by CMS commands and describes how they are used. Figure 3-2 on page 3-7 lists the filetypes used by CMS/DOS commands.

In addition to these CMS filetypes, there are special filetypes reserved for use by the language processors, which are IBM program products. These filetypes, and the commands that use them, are:

Filetypes	Commands
COBOL, SYMDMP, TESTCOB	COBOL, FCOBOL, TESTCOB
FORTRAN, FREEFORT, FTnn001, TESTFORT	FORTRAN, FORTGI, FORTHX, GOFORT, TESTFORT
PLI, PLIOPT	DOSPLI, PLIC, PLICR, PLIOPT
RPGII	RPGII
VSBASIC, VSBDATA	VSBASIC

For details on how to use these filetypes, consult the appropriate program product documentation.

Filetype	Command	Comments
AMSERV	AMSERV	Contains VSAM access method services control statements executed with the AMSERV command.
ASM3705	ASM3705 GEN3705	Used by system programmers to generate the 3704/3705 control program.
ASSEMBLE	ASSEMBLE	Contains source statements for assembler language programs.
AUXxxxx	UPDATE	Points to files that contain UPDATE control statements for multiple updates.
CNTRL	UPDATE	Lists files that either contain UPDATE control statements or point to additional files.
COPY	MACLIB	Can contain COPY control statements and macros or copy files to be added to MACLIBs.
DIRECT	DIRECT	Contains entries for the VM/SP user directory file.The system programmer controls this file.
EXEC	EXEC GEN3705 LISTFILE	Can contain sequences of CMS or user-written commands, with execution control statements.
GLOBALV	DEFAULTS GLOBALV	Contains variables used by GLOBALV.

Figure 3-1 (Part 1 of 3). Filetypes Used by CMS Commands

Filetype	Command	Comments
HELPCMS HELPCP HELPDEBU HELPEDIT HELPEXC2 HELPEXEC HELPHELP HELPMENU HELPMSG HELPPREF HELPREXX HELPSET HELPSQLD HELPXEDI	HELP	Contains descriptive information for CP and CMS commands, messages, Restructured Extended Executor (REXX), EXEC 2, and EXEC statements, CMS editor and System Product editor subcommands, and menu lists and the SQL/Data System Program Product (5748-XXJ) (only if you have this installed on your system.)
LISTING	AMSERV ASSEMBLE ASM3705 COBOL DOSPLI FCOBOL LOADLIB PLIOPT	Listings are produced by the assembler, the language processors, and the AMSERV and LOADLIB commands.
LKEDIT	LKED	Contains the printer output from the LINK EDIT of a CMS text file or OS object module.
LOADLIB	FILEDEF GLOBAL LKED LOADLIB NUCXLOAD OSRUN QUERY ZAP	Is a library created by the LKED command or the LOADLIB utility command. The GLOBAL or FILEDEF command identifies the libraries that should be searched for program execution. NUCXLOAD loads a member of a CMS LOADLIB library or an OS module library. OSRUN executes a member of a CMS LOADLIB library or an OS module library. Query indicates the libraries that were affected by the GLOBAL command. ZAP is used to modify an existing LOADLIB member.
MACLIB	GLOBAL MACLIB	Library members contain macro definitions or copy files; the MACLIB command creates the library, and lists, adds, deletes, or replaces members. The GLOBAL command identifies which macro libraries should be searched during an assembly or compilation.
MACRO	MACLIB	Contains macro definitions to be added to a CMS macro library (MACLIB).
MAP	INCLUDE LOAD MACLIB TAPE TXTLIB	Maps created by the LOAD and INCLUDE commands indicate entry point locations; the MACLIB, TXTLIB, and TAPE commands produce MAP files.

Figure 3-1 (Part 2 of 3). Filetypes Used by CMS Commands

Filetype	Command	Comments
MODULE	GENMOD LOADMOD MODMAP NUCXLOAD	MODULE files created by the GENMOD command are nonrelocatable executable programs. The LOADMOD commands loads a MODULE file for execution; the MODMAP command displays a map of entry point locations. NUCXLOAD loads a module into free storage and defines it as a nucleus extension.
NAMES	NAMEFIND NAMES	Contains information regarding users with whom you communicate.
NETLOG	RECEIVE SENDFILE	Contains records which log the transmission of files sent by or received by you.
NOTEBOOK	RECEIVE SENDFILE	Contains notes sent to you or sent by you to to other users.
SYNONYM	SYNONYM	Contains a table of synonyms for CMS commands and user-written EXEC and MODULE files.
SCRIPT	SCRIPT	SCRIPT text processor input includes data and SCRIPT control words.
TEXT	ASSEMBLE INCLUDE LOAD TXTLIB	TEXT files contain relocatable object code created by the assembler and compilers. The LOAD and INCLUDE commands load them into storage for execution. The TXTLIB command manipulates libraries of TEXT files.
TXTLIB	GLOBAL TXTLIB	Library members contain relocatable object code. The TXTLIB command creates the library, and lists or deletes existing members. The GLOBAL command identifies TXTLIBs to search.
UPDATE	UPDATE	Contains UPDATE control statements for single updates applied to source programs.
UPDLOG	UPDATE	Contains a record of additions, deletions, or changes made with the UPDATE command.
UPDTxxxx	UPDATE	Contains UPDATE control statements for multilevel updates.
ZAP	ZAP	Contains control records for the ZAP command, which is used by system support personnel.

Figure 3-1 (Part 3 of 3). Filetypes Used by CMS Commands

Filetype	Command	Comments
COPY	MACLIB SSERV	When the SSERV command copies books or macros from DOS source statement libraries, the output is written to CMS COPY files, which can be added to CMS macro libraries with the MACLIB command.
DOSLIB	DOSLIB DOSLNK FETCH GLOBAL	DOS core image phases are placed in a DOSLIB by linkage editor, invoked with the DOSLNK command. The GLOBAL command identifies DOSLIBs to be searched when the FETCH command is executed.
DOSLNK	DOSLKED	Contains linkage editor control statements for input to the CMS/DOS linkage editor.
ESERV	ESERV	Contains input control statements for the ESERV utility program.
EXEC	LISTIO	The LISTIO command with the EXEC option creates the $LISTIO EXEC that lists system and programmer logical unit assignments.
LISTING	ASSEMBLE ESERV	Listings contain processor output from the ESERV command, and compiler output from the assembler and language processors.
MACRO	ESERV MACLIB	Contains SYSPCH output from the ESERV program, suitable for addition to a CMS MACLIB file.
MAP	DOSLIB DOSLKED DSERV	The DSERV command creates listings of the directories of DOS libraries. The DOSLIB command with the MAP option produces a list of DOSLIB members. The linkage editor map from the DOSLKED command is written into a MAP file.
PROC	PSERV	The PSERV command copies procedures from DOS procedure libraries into CMS PROC files.
TEXT	ASSEMBLE DOSLKED RSERV	Object decks created by the assembler or compilers are written into TEXT files. The RSERV command creates TEXT files from modules in DOS relocatable libraries. TEXT files can also be used as input to the linkage editor.

Figure 3-2. Filetypes Used in CMS/DOS

Output Files: TEXT and LISTING

Output files from the assembler and the language processors are logically related to the source programs by their filenames. Some of these files are permanent and some are temporary. For example, if you issue the command:

```
assemble myfile
```

CMS locates a file named MYFILE with a filetype of ASSEMBLE and the system assembler assembles it. If the file is on your A-disk, then when the assembler completes execution, the permanent files you have are:

```
MYFILE ASSEMBLE A1
MYFILE TEXT     A1
MYFILE LISTING  A1
```

where the TEXT file contains the object code resulting from the assembly, and the LISTING file contains the program listing generated by the assembly. If any TEXT or LISTING file with the same name previously existed, it is erased. The source input file, MYFILE ASSEMBLE A1, is neither erased nor changed.

Because these files are CMS files, you can use the editor to examine or modify their contents. If you want a printed copy of a LISTING file, you can use the PRINT command to print it. If you want to examine a TEXT file, you can use the TYPE or PRINT command specifying the HEX option.

> **Note:** If a TEXT file contains control changes for the terminal, the edit lines may not be displayed in their true form. Therefore, it is suggested you do not use the editor for TEXT files, because the results are unpredictable. Instead, use the TYPE or PRINT commands with the HEX option to display TEXT decks. Put TEXT decks into a TXTLIB and ZAP the TXTLIB to modify the TEXT deck.

Filetypes for Temporary Files

The filetypes of files created by the assembler and language processors for use as temporary workfiles are:

SYSUT1	SYS001	SYS004
SYSUT2	SYS002	SYS005
SYSUT3	SYS003	SYS006
SYSUT4		

Figure 3-3. Filetypes for Temporary Work Files

CMS handles all SYSUTx and SYS00x files as temporary files.

The CMS AMSERV command, executing VSAM utility functions, uses two workfiles that have filetypes of LDTFDI1 and LDTFDI2.

The CMS RECEIVE command, when receiving files in DISK DUMP format, creates a temporary file with the fileid AAAA BBBB A.

Disk space is allocated for temporary files on an as-needed basis. They are erased when processing is complete. If a program you are executing is terminated before completion, these workfiles may remain on your disk. You can erase them.

CMSUT1 Files

The CMSUT1 filetype is used by CMS commands that create files on your CMS disks. The CMSUT1 file is used as a workfile and is erased when processing is complete. When a command fails to complete execution properly, the CMSUT1 file may not be erased. CMSUT1 files are reserved for system usage, and use of these files may cause unpredictable results. The commands, and the filenames they assign to files they create, are listed below.

Command	Filename
COPYFILE	COPYFILE
DISK LOAD	DISK
EDIT	EDIT
INCLUDE	DMSLDR
LOAD	DMSLDR
MACLIB	DMSLBM
READCARD	READCARD
TAPE LOAD	TAPE
UPDATE	fn (the filename of the UPDATE file)

Filetypes for Documentation

There are two CMS reserved filetypes for which the System Product editor and CMS editor accept (by default) uppercase and lowercase input data. These are MEMO and SCRIPT.

- You can use MEMO files to document program notes or to write reports.
- The SCRIPT filetype is used by the SCRIPT command. This command invokes a text processor that is part of the IBM Document Composition Facility program product.

Filemode Letters and Numbers

The filemode field of a CMS file identifier has two characters: the filemode letter and the filemode number.

- The filemode letter is established by the ACCESS command and specifies the virtual disk on which a file resides: A through Z.
- The filemode number is a number from 0 to 6, which you can assign to the file when you create it or rename it; if you do not specify it, the value defaults to 1.

How you access your disks and what filemode letters you give them with the ACCESS command depends on how you want to use the files that are on them.

For most of the reading and writing you do of files, you use your A-disk, which is also known as your primary disk. This is a read/write disk. You may access other disks in your configuration, or access linked-to disks, in read-only or read/write status, depending on whether you have a read-only or read/write link.

When you load CMS (with the IPL command), your virtual disk at address 191 is accessed for you as your A-disk. Your virtual disk at address 190 (the system disk) is accessed as your S-disk; and the disk at 19E is accessed as an extension of your S-disk, with a mode letter of Y. The S-disk and Y-disk are accessed for only mode S2 and Y2 files, thus:

```
access 190 S/S * * S2
access 19E Y/S * * Y2
```

In addition, if you have a disk defined at address 192, it is accessed for you as your D-disk. If the 192 disk has not been formatted, CMS will do it automatically and label the minidisk 'SCRTCH'.

If ACCESS is the first command issued after an IPL of the CMS system, only the A-disk is not automatically defined. Another ACCESS command must be issued to define the A-disk.

The actual letters you assign to any other disks (and you may reassign the letters A, D, and Y), is arbitrary; but it does determine the CMS search order, which is the order in which CMS searches your disks when it is looking for a file. The order of search (when all disks are being searched) is alphabetical: A through Z. If you have duplicate file identifiers on different disks, you should check your disk search order before issuing commands against that filename to be sure that you will get the file you want. You can find out the current search order for your virtual disks by issuing the command:

```
query search
```

You can also access disks as logical extensions of other disks, for example:

```
access 235 b/a
```

The "/A" indicates that the B-disk is to be a read-only extension of the A-disk, and the A-disk is considered the "parent" of the B-disk. A disk may have many extensions, but only one level of extension is allowed. If you access an extension A-disk containing no files, the access fails.

How Extensions Are Used

If you have a disk accessed as an extension of another disk, the extension disk is automatically read-only, and you cannot write on it. You might access a disk as its own extension, therefore, to protect the files on it, so that you do not accidentally write on it. For example:

```
access 235 b/b
```

Another use of extensions is to extend the CMS search order. If you issue a command requesting to read a file, for example:

```
type alpha plan
```

CMS searches your A-disk for the file named ALPHA PLAN and if it does not find it, searches any extensions that your A-disk may have. If you have a file named ALPHA PLAN on your B-disk but have not accessed it as an extension of your A-disk, CMS will not find the file, and you will have to reenter the command:

```
type alpha plan b
```

Additionally, if you issue a CMS command that reads and writes a file, and the file to be read is on an extension of a read/write disk, the output file is written to the parent read/write disk. The XEDIT command is a good example of this type of command. If you have a file named FINAL LIST on a B-disk extension of a read/write A-disk, and if you invoke the editor to modify the file with the command:

```
xedit final list
```

after you have made modifications to the file, the changed file is written onto your A-disk. The file on the B-disk remains unchanged.

Accessing and Releasing Read-Only Extensions

When you access a disk as a read-only extension, it remains an extension of the parent disk as long as both disks are still accessed. If either disk is released, the relationship of parent disk/extension is terminated.

If the parent disk is released, the extension remains accessed and you may still read files on it. If you access another disk at the mode letter of the original parent disk, the parent/extension relationship remains in effect.

If you release a read-only extension and access another disk with the same mode letter, it is not an extension of the original parent disk unless you access it as such. For example, if you enter:

```
access 198 c/a
release c
access 199 c
```

the C-disk at virtual address 199 is not an extension of your A-disk.

When to Specify Filemode Letters: Reading Files

When you request CMS to access a file, you have to identify it so that CMS can locate it for you. The commands that expect files of particular filetypes (reserved filetypes) allow you to enter only the filename of the file when you issue the command. When you execute any of these commands or execute a MODULE or EXEC file, CMS searches all of your accessed disks (using the standard search order) to locate the file. Some CMS commands that perform this type of search are:

AMSERV	GLOBAL	MODMAP
ASSEMBLE	LOAD	RUN
DOSLIB	LOADMOD	TXTLIB
EXEC	MACLIB	

Some CMS commands require you to enter the filename and filetype to identify a file. You may specify the filemode letter; if you do not specify the filemode, CMS searches only your A-disk and its extensions when it looks for the file. If you do specify a filemode letter, the disk you specify and its extensions are searched for the file. Some commands you can use this way are:

EDIT	PUNCH	TAPE
FILEDEF	STATE	TYPE
PRINT	SYNONYM	UPDATE

There are some CMS commands that do not search extensions of disks when looking for files. They include:

```
ERASE
FILELIST
LISTFILE
```

You must explicitly enter the filemode if you want to use these commands to list or dump files that are on extensions.

The following commands search every accessed read-only and read-write disk.

```
NAMES
NAMEFIND
```

Using Asterisks and Equal Signs

For some CMS commands, if you specify the filemode of a file as an asterisk, it indicates that you either do not know or do not care what disk the file is on and you want CMS to locate it for you. For example, if you enter:

```
listfile myfile test *
```

the LISTFILE command responds by listing all files on your accessed disks named MYFILE TEST. When you specify an asterisk for the filemode of the COPYFILE, ERASE, or RENAME commands, CMS locates all copies of the specified file. For example:

```
rename temp sort * good sort =
```

renames all files named TEMP SORT to GOOD SORT on all of your accessed read/write disks. An equal sign (=) is valid in output fileids for the RENAME and COPYFILE commands.

For some commands, when you specify an asterisk for the filemode of a file, CMS stops searching as soon as it finds the first copy of the file. For example:

```
type myfile assemble *
```

If there are files named MYFILE ASSEMBLE on your A-disk and C-disk, then only the copy on your A-disk is displayed. The commands that perform this type of search are:

```
COMPARE          PRINT          STATE
DISK DUMP        PUNCH          SYNONYM
EDIT             RUN            TAPE DUMP
FILEDEF          SORT           TYPE
```

For the COMPARE, COPYFILE, RENAME, and SORT commands, you must always specify a filemode letter, even if it is specified as an asterisk.

When to Specify Filemode Letters: Writing Files

When you issue a CMS command that writes a file onto one of your virtual disks, and you specify the output filemode, CMS writes the file onto that disk. The commands that require you to specify the output filemode are:

```
COPYFILE
RENAME
SORT
```

The commands that allow you to specify the output filemode, but do not require it, are:

```
FILEDEF          TAPE LOAD
GENMOD           TAPPDS
READCARD         UPDATE
```

When you do not specify the filemode on these commands, CMS writes the output files onto your A-disk.

Some CMS commands that create files always write them onto your A-disk. The LOAD and INCLUDE commands write a file named LOAD MAP A5. The

LISTFILE command creates a file named CMS EXEC, on your A-disk. The CMS/DOS commands DSERV, ESERV, SSERV, PSERV, and RSERV also write files onto your A-disk.

Other commands that do not allow you to specify the filemode, write output files either:

- To the disk from which the input file was read, or
- To your A-disk, if the file was read from a read-only disk

These commands are:

```
AMSERV
MACLIB
TXTLIB
UPDATE
```

The SORT command also functions this way if you specify the output filemode as an asterisk (*).

In addition, many of the language processors, when creating work files and permanent files, write onto the first read/write disk in your search order, if they cannot write on the source file's disk or its parent.

How Filemode Numbers are Used

Whenever you specify a filemode letter to reference a file, you can also specify a filemode number. Since a filemode number for most of your files is 1, you do not need to specify it. The filemode numbers 0 through 6 are discussed below. Filemode numbers 7 through 9 are reserved for IBM use.

Filemode 0

A filemode number of 0 assigned to a file makes that file private. No other user may access it unless they have read/write access to your disk. Under normal circumstances; if someone links to your disk in read-only mode and requests a list of all the files on your disk, the files with a filemode of 0 are not listed.

The DDR command will allow you to copy the minidisk from one disk to another, and therefore, the filemode 0 files. Use a read share password to protect minidisks with private files when using ACCESS.

Filemode 1

Filemode 1 is used for reading and writing files. It is the default filemode.

Filemode 2

Filemode 2 is essentially the same, for the purposes of reading and writing files, as filemode 1. Usually a filemode of 2 is assigned to files that are shared by users who link to a common disk, like the system disk. Since you can access a disk and specify which files on that disk you want to access, files with a filemode of 2 provide a convenient subset of all files on a disk. For example, if you issue the command:

```
access 489 e/a * * e2
```

you can only read files with a filemode of 2 on the disk at virtual address 489.

Filemode 3

Files with a filemode of 3 are erased after they are read. If you create a file with a filemode of 3 and then request that it be printed, the file is printed, and then erased. You can use this filemode if you write a program or EXEC procedure that creates files that you do not want to maintain copies of on your virtual disks. You can create the file, print it, and not have to worry about erasing it later.

The language processors and some CMS commands create work files and give these work files a filemode of 3.

> **Note:** A filemode of 3 should not be used with EXECs. Depending on what commands are issued within it, an EXEC with a filemode of 3 may be erased before it completes execution.

Filemode 4

Files with a filemode of 4 are in OS simulated data set format. These files are created by OS macros in programs running in CMS. You specify that a file created by a program is to have OS simulated data set format by specifying a filemode of 4 when you issue the FILEDEF command for the output file. If you do not specify a filemode of 4, the output file is created in CMS format.

You can find more details about OS simulated data sets in Chapter 9, "Developing OS programs under CMS" on page 9-1.

> **Note:** There are no filemode numbers reserved for DOS or VSAM data sets, since CMS does not simulate these file organizations.

Filemode 5

This filemode number is the same, for purposes of reading and writing, as filemode 1. You can assign a filemode of 5 to files that you want to maintain as logical groups, so that you can manipulate them in groups. For example, you can reserve the filemode of 5 for all files that you are retaining for a certain period of time; then, when you want to erase them, you could issue the command:

```
erase * * a5
```

Filemode 6

The filemode number 6 indicates that the "update-in-place" attribute of a CMS file is in effect. This means that the existing records of a file are written back to their previous location on disk rather than in a new slot. This only applies to files located on 512-, 1K-, 2K-, or 4K-byte block formatted minidisks. To take advantage of the "update-in-place" capability, the FSWRITE macro must be used, whether explicitly by the user or implicitly by the system.)

> **Note:** For a variable format file, "update-in-place" applies only if a record is replaced by a record of the same length.

When To Enter Filemode Numbers

You can assign filemode numbers when you use the following commands:

COPYFILE — You can assign a filemode number when you create a new file with the COPYFILE command.

DLBL, FILEDEF — When you assign file definitions to disk files for programs or CMS command functions, you can specify a filemode number.

GENMOD — You can specify a filemode number on the GENMOD command line. To change the filemode number of an existing MODULE file, use the RENAME or COPYFILE commands.

READCARD — You can assign a filemode number when you specify a file identifier on the READCARD command line or on a READ control card.

RECEIVE — You can assign a filemode number when receiving a file from your virtual reader.

RENAME — When you specify the fileids on the RENAME command, you can specify the filemode numbers for the input and/or output files. To change only the filemode number of an existing file, you must use the RENAME command. For example:

```
RENAME test module a1 = = a2
```

changes the filemode number of the file TEST MODULE A from 1 to 2.

SORT — You can specify filemode numbers for the input and/or output fileids on the SORT command line.

XEDIT — You can assign a filemode number when you create a file with the System Product editor. To change the filemode number of an existing file, use the RENAME or COPYFILE commands, or use the SET FMODE subcommand when you are in the edit environment.

Managing Your CMS Disks

The number of files you can write on a CMS disk depends on both the size of the disk and the size of the files that it contains. You can find out how much space is being used on a disk by using the QUERY DISK command. For example, to see how much space is on your A-disk, you would enter:

```
query disk a
```

The response may be something like this:

```
LABEL  cuu  M  STAT  CYL  TYPE  BLKSIZE  FILES  BLKS USED-(%)  BLKS LEFT  BLK TOTAL
MYDISK 191  A  R/W    5  3330   1024      171       1221-92        107       1328
```

When a disk is becoming full, you should erase whatever files you no longer need, or dump to tape files that you need to keep but do not need to keep active on disk.

When you are executing a command or program that writes a file to disk, and the disk becomes full in the process, you receive an error message, and you have to try to clear some space on the disk before you can attempt to execute the command or program again. To avoid the delays that such situations cause, you should try to maintain an awareness of the usage of your disks. If you cannot erase any more files from your disks, you should contact installation support personnel about obtaining additional read/write CMS disk space.

CMS File Directories

Each CMS disk has a master file directory that contains entries for each of the CMS files on the disk. When you access a disk, information from the master file directory is brought into virtual storage and written into a user file directory. The user file directory has an entry for each file that you may access. If you have accessed a disk specifying only particular files, then the user file directory contains entries only for those files.

If you have read/write access to a disk, then each time you write the file onto disk the user file directory and master file directory are updated to reflect the current status of the disk. If you have read/write access to a disk and the FSCLOSE macro is issued, the user file directory is updated. When there are no open files on the disk, the master file directory is updated to reflect the current status of the files. If you have read-only access to a disk, then you cannot update the master file directory or user file directory. If you access a read-only disk while another user is writing files onto it, you may need to periodically reissue the ACCESS command for the disk, to obtain a fresh copy of the master file directory.

> **Note:** You should never attempt to write on a disk at the same time as another user.

The user file directory remains in virtual storage until you issue the RELEASE command specifying the mode letter or virtual address of the disk. If you detach a virtual disk (with the CP DETACH command) without releasing it, CMS does not know that the disk is no longer part of your virtual machine. When you attempt to read or write a file on the disk CMS assumes that the disk is still active (because the user file directory is still in storage) and encounters an error when it tries to read or write the file.

A similar situation occurs if you detach a disk and then add a new disk to your virtual machine using the same virtual address as the disk you detached. For example, if you enter the following sequence of commands:

```
cp link user1 191 195 rr rpass
access 195 d
cp detach 195
cp link user2 193 195 rr rpass
listfile * * d
```

the LISTFILE command produces a list of the files on USER1's 191 disk; if you attempt to read one of these files, you receive an error message. You must issue the ACCESS command to obtain a copy of the master file directory for USER2's 193 disk.

Note: The password cannot be entered on the command line if the password suppression facility was specified when your system was installed.

The entries in the master file directory are sorted alphamerically by filename and filetype, to facilitate the CMS search for particular files. When you are updating disk files, the entries in the user file directory and master file directory tend to become unsorted as files are created, updated, and erased. When you use the RELEASE command to release a read/write disk, the entries are sorted and the master file directory is rewritten. If you or any other user subsequently access the disk, the file search may be more efficient.

CMS Command Search Order

When you enter a command line in the CMS environment, CMS has to locate the command to execute. If you have EXEC or MODULE files on any of your accessed disks, CMS treats them as commands; also, they are known as user-written commands.

As soon as the command name is found, the search stops and the command is executed. The search order is:

1. Search for a file with filetype EXEC on any currently accessed disk. CMS uses the standard search order (A through Z.)
2. Search for a valid name on any currently accessed disk, according to current SYNONYM file definitions in effect.
3. Search for a nucleus extension command if the high order byte of register 1 is not equal to X'03' or X'04'.
4. Search for a command in the transient area.
5. Search for a nucleus-resident command.
6. Search for a file with filetype MODULE on any currently accessed disk.
7. Search for a valid abbreviation or truncation of a nucleus extension.
8. Search for a valid abbreviation or truncation of a command in the transient area.
9. Search for a valid abbreviation or truncation of a command in the nucleus.
10. Search for a valid abbreviation or truncation of any other CMS command.
11. Search for a CP command.
12. Search for a valid abbreviation or truncation of a CP command.

For example, if you create a command module that has the same name as a CMS nucleus-resident command, your command module cannot be executed, since CMS locates the nucleus-resident command first, and executes it. When a user-written command has the same name as a CMS command module abbreviation, certain error messages may indicate the CMS command name, rather than the program name.

Figure 3-4 illustrates details of the command search order.

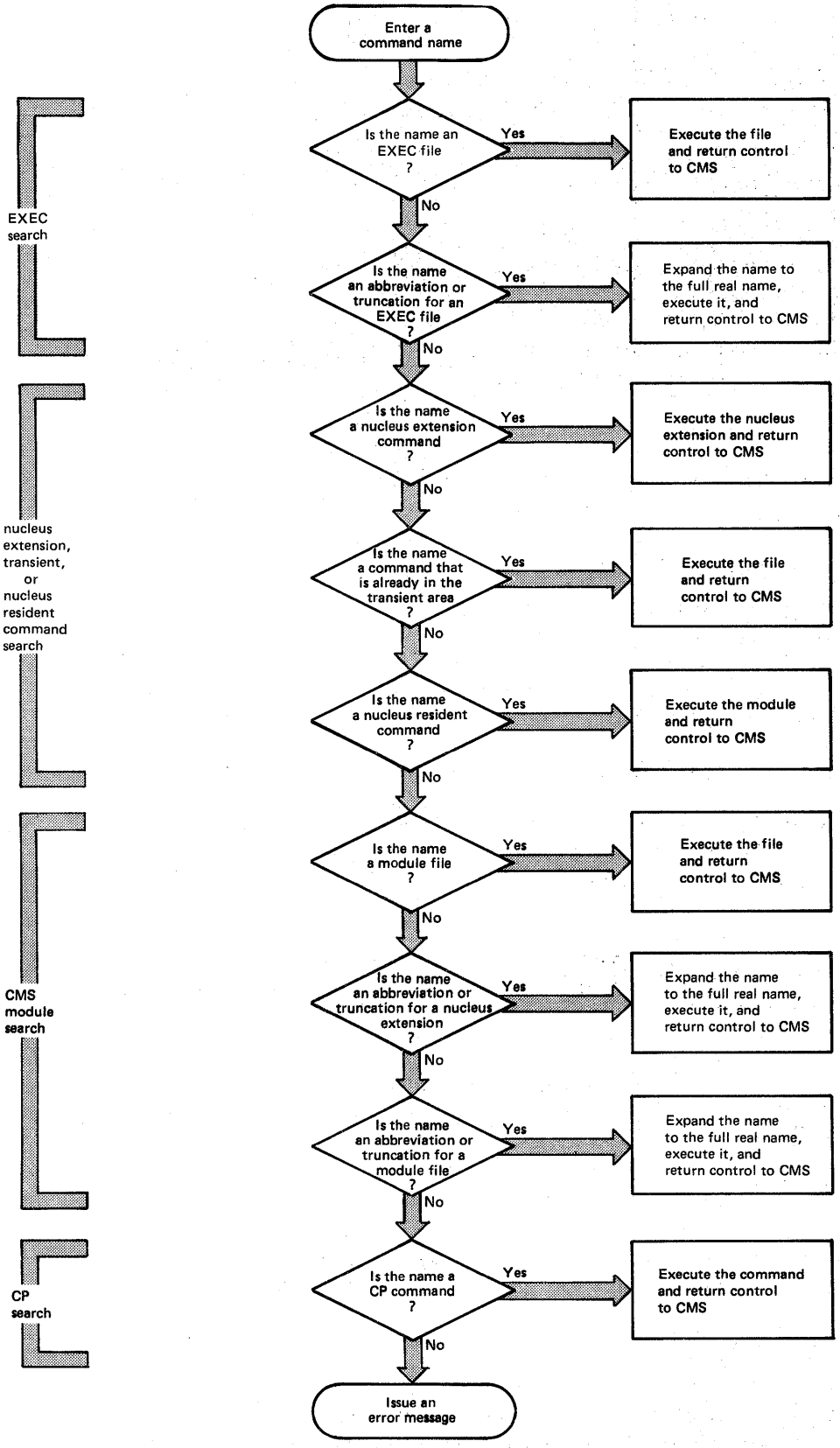


Figure 3-4. How CMS Searches for the Command to Execute

CMS Command Execution Characteristics

Following is an alphabetical list of the CMS commands and their execution characteristics. The "Code" column indicates the execution characteristics of the command.

Code	Meaning
E	indicates that this command is an EXEC.
N	indicates that this command executes in the nucleus or is a nucleus extension.
S	indicates that this command issues the STRINIT macro, causing a reset of the OS free storage pointers.
T	indicates that this command executes in the transient area.
U	indicates that this command executes in the user program area.

Figure 3-5. CMS Command Execution Characteristics

Command	Code	Command	Code
ACCESS	T	FILELIST	E
AMSERV	U	FINIS	N
ASSEMBLE	U	FORMAT	U
ASSGN	T	GENDIRT	T
CATCHECK	U	GENMOD	N
CMDCALL	N	GLOBAL	T
CMSBATCH	U	GLOBALV	T
COMPARE	T	HELP	N,S
CONWAIT	N	IDENTIFY	T
COPYFILE	U,S	IMMCMD	N
CP	N	INCLUDE	N
DDR	U,S	IOCP	U
DEBUG	N	LABELDEF	T
DEFAULTS	E	LISTDS	U
DESBUF	N	LISTFILE	N
DISK	T	LISTIO	T
DLBL	T	LKED	U
DOSLIB	U	LOAD	N
DOSLKED	U	LOADLIB	U,S
DOSPLI	E,U	LOADMOD	N
DROPBUF	N	MACLIB	U
DSERV	U	MAKEBUF	N
EDIT	U,S	MODMAP	U
ERASE	N	MOVEFILE	U,S
ESERV	E	NAMEFIND	N
EXEC	N	NAMES	E
EXECIO	N	NOTE	E
EXECOS	N	NUCXDROP	T
EXECUPDT	E	NUCXLOAD	T
FCOBOL	E,U	NUCXMAP	T
FETCH	N	OPTION	T
FILEDEF	T	OSRUN	U

Command	Code
PEEK	E
PRINT	T
PSERV	U
PUNCH	T
QUERY	N
RDR	T
RDRLIST	E
RECEIVE	E
READCARD	T
RELEASE	T
RENAME	N
RESERVE	T
RSERV	U
RUN	E
SENDFILE	E
SENTRIES	N
SET	T,S
SETPRT	T
SORT	U,S
SSERV	U
START	N
STATE	N
STATEW	N
SVCTRACE	T
SYNONYM	T
TAPE	T
TAPEMAC	U
TAPPDS	U
TELL	E
TXTLIB	U
TYPE	T
UPDATE	U
XEDIT	N,S

Displaying a List of Your CMS Files

Use the FILELIST command to display information about your CMS files residing on accessed disks. In a full screen environment, FILELIST provides you with the same information as the LISTFILE command, but also allows you to edit and issue commands from the list. You can issue XEDIT subcommands to manipulate the list itself. Figure 3-6 is a sample FILELIST list.

```
  ZOOKEEP FILELIST     A1  V 108  TRUNC=108 SIZE=418 LINE=1 COL=1 ALT=0
Cmd Filename Filetype Fm Format Lrecl Records Blocks   Date    Time
    ALL      NOTEBOOK A2 V         120     277     10  9/24/82  9:14:02
    ANIMAL   DATA     A1 V          95      34      2 10/04/82 21:12:04
    BANANA   DATA     A1 V          95      29      2 10/04/82 20:58:07
    BEAR     NOTE     A1 V         107     281     10 10/04/82 17:59:00
    HONEY    DATA     A1 V          92     101      4 10/02/82 15:33:05
    LION     NOTE     A2 V          75      28      1  9/25/82 12:10:03
    TIGER    NOTE     A1 V          26       7      1  9/23/82 16:50:06
    ZOOKEEP  NETDATA  A1 V          80     489     30  8/26/82 16:05:08
1= Help     2= Refresh  3= Quit    4= Sort(type) 5= Sort(date) 6= Sort(size)
7= Backward 8= Forward  9= FL /n  10=           11= XEDIT     12= Cursor

====> _
                                                                  XEDIT
```

Figure 3-6. Sample FILELIST Screen

Finding Files in Your FILELIST List

If you have many files in your list, the list may take up more than one screen. To find files in your FILELIST list, you can do any of the following:

- Scroll through the list using the PF keys.

Key	Function
PF7	Scrolls backward one full screen.
PF8	Scrolls forward one full screen.

- Rearrange the list using one of the following keys:

Key	Arrangement
PF4	Orders the list by filetype.
PF5	Orders the list by date (newest to oldest). This is how the list is initially arranged.
PF6	Orders the list by size (largest to smallest).

- Use the XEDIT subcommand LOCATE if you know the filename and/or filetype of the file that you are looking for. You enter the LOCATE command at the bottom of the screen and then press the ENTER key. For example:

  ```
  ====> locate/banana data/
  ```

  If BANANA DATA is located, the line containing it becomes the first line on the screen.

- Rearrange the list by entering one of the following synonyms on the command line.

SNAME	Sorts the list alphabetically by filename, filetype, and filemode.
STYPE	Sorts the list alphabetically by filetype, filename, and filemode.
SMODE	Sorts the list by filemode, filename, and filetype.
SRECF	Sorts the list by record format, filename, filetype, and filemode.

SLREC	Sorts the list by logical record length and then by size (greatest to least).
SSIZE	Sorts the list by number of blocks and number of records (greatest to least).
SDATE	Sorts the list by year, month, day, and time (most recent to oldest).

Using FILELIST to List Some of Your Files

FILELIST allows you to obtain various lists of the files on your disks. You can ask for a list of files that have the same filename or filetype or all of the files that begin with a certain letter. The abbreviation for FILELIST is FILEL. Following are various ways that you might use the FILELIST command:

filel	Displays a lists of the files on your A-disk.
filel * * b	Displays a lists of the files on your accessed B-disk.
filel bear *	Displays a list of of the files on your A-disk with a filename of BEAR.
filel * data	Displays a list of files on your A-disk with DATA as the filetype.
filel * * a1	Displays a list of the files with a filemode number 1 on your A-disk.

Erasing Files from FILELIST

Use the DISCARD command to erase from disk a file that is displayed in the list. DISCARD is equivalent to the CMS command ERASE. DISCARD can either be typed in the command area of the line that describes the file you want discarded, or it can be entered from the command line (at the bottom of the screen). DISCARD can only be used while in FILELIST, RDRLIST, and PEEK commmand environments.

Listing your Files with the LISTFILE command

You can use the LISTFILE command to list information about your CMS disk files. For example, entering:

```
listfile * data
```

lists the files on your A-disk with the filetype of DATA. For example:

```
ANIMAL   DATA     A1
BANANA   DATA     A1
HONEY    DATA     A1
```

If you want more information than just the fileids, you can use one of the information request options for LISTFILE. For example, entering:

```
listfile * data (label
```

returns a list with more than just the filelid. For example:

```
FILENAME FILETYPE FM FORMAT LRECL RECS  BLOCKS   DATE      TIME     LABEL
ANIMAL   DATA     A1 V          95   34       2  10/04/82 21:12:04 ZKP191
BANANA   DATA     A1 V          95   29       2  10/04/82 20:58:07 ZKP191
HONEY    DATA     A1 V          92  101       4  10/02/82 15:33:05 ZKP191
```

As with the FILELIST command, you can vary what you list with the LISTFILE command. Remember you only need to enter L, the minimum truncation for LISTFILE. Following, are various ways that you might use the LISTFILE command:

l	Lists the files on your A-disk.
l * * b	Lists the files on your accessed B-disk.
listf bear *	Lists the files on your A-disk with a filename of BEAR.
l * data	Lists the files on your A-disk with DATA as a filetype.
list * * a1	Lists the files with a filemode number 1 on your A-disk.

Comparing Contents of Files

To compare the contents of two files to see if they are identical, use the COMPARE command. For example:

```
compare labor stat a1 labor stat b1
```

Any records in these files that do not match are displayed at your terminal. The format of the COMPARE command is documented in the *VM/SP CMS Command and Macro Reference*.

Copying Files

The COPYFILE command, in its simplest form, copies a file from one virtual disk to another. For example:

```
copyfile linda assemble b pat assemble a
```

Renaming Files

You can change the file identifier of a file with the RENAME command.

```
rename test file a1 good file a1
```

You can use RENAME to modify filemode numbers, for example:

```
rename * module a1 = = a2
```

changes the filemode numbers on all MODULE files on the A-disk that have a mode number of 1 to a mode number of 2. Remember that you cannot use RENAME to move a file from one disk to another. You must use the COPYFILE command to change filemode letters.

Using Synonyms

By using the SYNONYM and the SET ABBREV commands, you can control what command names, synonyms, or truncations are valid in CMS. For example, you could create a file named MYSYN SYNONYM which contains the following records:

```
PRINT    PRT    1
RELEASE  LETGO  4
FILELIST FL     2
```

The first column specifies an existing CMS command, module, or EXEC name. The second column specifies the alternate name or synonym that you want to use. The third column is a count field that indicates the minimum number of characters of the synonym that can be used to truncate the name. Using this file, after you enter the command:

```
synonym mysyn
```

you can use PRT, LETGO, and FL in place of the corresponding CMS command names. Also, if the ABBREV function is in effect, (it is the default; you can make sure it is in effect by issuing the command SET ABBREV ON), you can truncate any of your synonyms to the minimum number of characters specified in the count field of the record (that is, you could enter P for PRINT and LETG for RELEASE). To invoke your synonym table at the beginning of every terminal session, enter the SYNONYM MYSYN command (or your own synonym table name) into your PROFILE EXEC.

Note: An EXEC procedure having a synonym defined for it can be invoked by its synonym if implied EXEC (IMPEX) function is on. However, within an EXEC procedure, only the EXEC filename can be used. A synonym is not recognized within an EXEC because the synonym tables are not searched during EXEC processing.

Chapter 4. What You Can Do with CMS Commands

This section provides an overview of some of the operations you might need to perform. The commands are not presented in their entirety, nor is a complete selection of commands represented.

As you glance through this section you should have an understanding of the kinds of commands available to you, so that when you need to perform a particular task using CMS you may have an idea of whether it can be done, and know what command to reference for details.

For complete lists of the CP and CMS commands available, see Figure D-1 and Figure E-2.

Beginning and Ending Your Terminal Session

Your terminal session starts when you logon (with CP LOGON) and ends when you logoff (CP LOGOFF). When you know that you are only going to be away from your terminal for a short while, you can disconnect (CP DISCONN). When you reconnect (with CP LOGON) the status of your virtual machine is the same as you left it.

Task	Command(s)	Description
Begin your terminal session	CP LOGON	Chapter 1
End your terminal session	CP LOGOFF	Chapter 1
	CP DISCONN	Chapter 1

The command formats and usage notes for the commands; DISCONN, LOGOFF, and LOGON, are documented in the *VM/SP CP Command Reference for General Users*.

Tailoring Your System

At the start of every terminal session you can automatically customize your system with the commands invoked by your PROFILE EXEC. Your PROFILE EXEC can include commands to set your PF keys, access disks, and access your synonym table.

If you are prone to typing errors, the RETRIEVE function provides you with a method of correcting errors without retyping the entire input. The RETRIEVE function is assigned to a PF key so that when the key is pressed, your previously entered line is retrieved.

When you are communicating with others on your computer, use the NAMES command to assign nicknames. The nicknames can be used with the SENDFILE and TELL commands, because both commands reference your NAMES file. You can create your own command to execute a series of commands by writing an EXEC.

Action	Command(s)	Description
Keep information about others with whom you communicate	NAMES	Chapter 7
Set your program function (PF) keys	CP SET PFnn	Chapter 1
Assign a PF key to retrieve previously entered lines	CP SET PFnn RETRIEVE	Chapter 1
Specify defaults for the commands: FILELIST, NOTE, PEEK, RDRLIST, RECEIVE, and SENDFILE	DEFAULTS	Chapter 7
Assign synonyms for system and your own commands	SYNONYM	Chapter 3
Tailor your System at the start of every session via your PROFILE EXEC	XEDIT and the System Product interpreter	Chapter 16
Write your own command that executes several commands or programs	XEDIT and the System Product interpreter	Chapter 15
Create your own Immediate command	IMMCMD	Chapter 8

The CP SET command is documented in the *VM/SP CP Command Reference for General Users*. The command formats and usage notes for the following CMS commands:

```
DEFAULTS
IMMCMD
NAMES
SYNONYM
XEDIT
```

are documented in the *VM/SP CMS Command and Macro Reference*.

Requesting Information

You can use CP and CMS commands to inquire about your terminal, virtual machine, disks, data files, and other users.

Inquiring about:	command(s)	Description
Terminal characteristics	CP QUERY SCREEN	Chapter 1
	CP QUERY SET	Chapter 1
	CP QUERY TERMINAL	Appendix C
Files on your disk	FILELIST	Chapter 3
	LISTFILE	Chapter 3
Files on your OS or DOS disks	LISTDS	Chapter 9
Files in your reader	RDRLIST	Chapter 7
	RDR	Chapter 7
	CP QUERY RDR ALL	Chapter 7
Your spool files	CP QUERY FILES	Chapter 7
Your virtual disks	QUERY DISK	Chapter 3
	QUERY SEARCH	Chapter 3
Your virtual machine	CP QUERY VIRTUAL STORAGE	Chapter 13
	IDENTIFY	Chapter 18
Your print files	CP QUERY PRINTER	Chapter 6
Other users	CP QUERY userid	Chapter 7
Your reader, printer, and punch	CP QUERY UR	Chapter 6

The command format and usage notes for the CP QUERY command are found in the *VM/SP Command Reference for General Users*. Command formats and usage notes for the following CMS commands can be found in the *VM/SP CMS Command and Macro Reference*.

```
FILELIST
IDENTIFY
LISTDS
LISTFILE
QUERY
RDR
RDRLIST
```

Communicating with Other Computer Users

You can use CP and CMS commands to send files, notes and messages to one or more users on your system or a system that is attached to yours via Remote Spooling Communications Subsystem (RSCS) network. The NOTE, SENDFILE, and TELL commands reference your "userid NAMES" file. The names file, created with the NAMES command, contains a collection of information about other computer users with whom you communicate.

Action	Commands	Description
Creating Names file	NAMES	Chapter 7
Sending files	SENDFILE	Chapter 7
	CP SPOOL, CP TAG, DISK DUMP	Chapter 7
	CP SPOOL, CP TAG, PUNCH	Chapter 9
Sending messages	TELL	Chapter 7
	CP MESSAGE	Chapter 7
Sending notes	NOTE and SENDFILE	Chapter 7

The command formats and usage notes for the following CP commands:

```
MESSAGE
SPOOL
TAG
```

can be found in the *VM/SP CP Command Reference for General Users.*

The following commands are CMS commands.

```
DISK DUMP
NAMES
NOTE
PUNCH
SENDFILE
TELL
```

Their command formats and usage notes are documented in the *VM/SP CMS Command and Macro Reference.*

Controlling Terminal Output

VM/SP allows you to control your terminal output. You can refuse messages with the CP SET MSG command. If you only want to see the short form of the CMS ready message, you set this with the CMS SET RDYMSG command.

The CP SCREEN command allows you to select colors and extended highlighting when you have the Extended Highlight feature and the Seven-Color feature on certain 3279 Models.

If your program is directing output to your terminal, you can halt the terminal display with the HT Immediate command, and later resume terminal display with the RT Immediate command.

Action	Command(s)	Description
Alter any extended color or highlighting definitions	CP SCREEN	Chapter 1
Control whether or not you receive messages	CP SET	Chapter 1
Indicate the type of CMS ready message that you want	SET RDYMSG	Chapter 1
Suppress terminal output	HT Immediate command	Chapter 2
Resume terminal output that was previously suppressed via HT	RT Immediate command	Chapter 2

The command formats for the CP commands:

```
SCREEN
SET
```

are documented in the *VM/SP CP Command Reference for General User's*. The command formats for the CMS commands:

```
HT (Immediate command)
RT (Immediate command)
SET
```

are documented in the *VM/SP CMS Command and Macro Reference*.

Sharing Virtual Disks

VM/SP allows you to share virtual disks on either a permanent or temporary basis. You can add another user's disk to your configuration with the CP LINK command. When you no longer need a disk that you have linked to or have temporarily accessed, you can release it with the CMS RELEASE command. When you no longer need a disk in your virtual machine configuration, you can disconnect it with the CP DETACH command.

Action	Command(s)	Description
Establish a link to a disk	CP LINK	Chapter 1
	ACCESS	Chapter 1
Release a disk	RELEASE	Chapter 1
	CP DETACH	Chapter 1

The formats and usage notes for the following CP commands:

```
DETACH
LINK
```

are documented in the *VM/SP CP Command Reference for General Users.*

The formats and usage notes for the following CMS commands:

```
ACCESS
RELEASE
```

are documented in the *VM/SP CMS Command and Macro Reference.*

Creating and Editing Files

Two editors are provided for you to create and modify files.

Editor	Command(s)	Description
System Product editor	XEDIT	Chapter 5
CMS editor	EDIT	Chapter 5 Appendix A

Complete information about the System Product editor is found in the *VM/SP System Product Editor Command and Macro Reference* and the *VM/SP System Product Editor User's Guide*. Refer to the *VM/SP CMS Command and Macro Reference* for information on EDIT subcommands and macros.

What You Can Do to the Files in Your Virtual Reader

CMS and CP commands allow you to look at, get rid of, keep, load onto disk, and reorder the files in your virtual reader.

Action	Command(s)	Description
Look at a file	PEEK	Chapter 7
Load the file onto your disk	RECEIVE	Chapter 7
	DISK LOAD	Chapter 7
	READCARD	Chapter 6
Purge a file	DISCARD (when in PEEK or RDRLIST)	Chapter 7
	CP PURGE	Chapter 7
Transfer a file to (or from) the reader queue of another user	CP TRANSFER	Chapter 7
Alter the external attributes of a file	CP CHANGE	Chapter 6
Change the order of the files	CP ORDER	Chapter 6

The CMS commands:

```
DISCARD
DISK LOAD
PEEK
READCARD
RECEIVE
```

are documented in the *VM/SP CMS Command and Macro Reference.*

The CP commands:

```
CHANGE
ORDER
PURGE
TRANSFER
```

are documented in the *VM/SP CP Command Reference for General Users.*

Receiving or Loading Files onto Your Disk

Files that are in your reader or on a tape can be loaded onto your disk.

Retrieving files from ...	Command(s)	Description
Your virtual reader	RECEIVE	Chapter 7
	DISK LOAD	Chapter 7
	READCARD	Chapter 6,7
A tape	TAPE LOAD	Chapter 6
	FILEDEF and MOVEFILE	Chapter 6

Command formats and usage notes for the following CMS Commands are documented in the *VM/SP Command and Macro Reference*.

```
DISK LOAD
FILEDEF
MOVEFILE
READCARD
RECEIVE
TAPE LOAD
```

Erasing Files from Your Virtual Disk

When you no longer need a file you can erase or discard it from your disk. You can use the ERASE command when you want to erase all the files with a particular filemode letter and number or the files with the same filename or filetype. You can use the DISCARD command when in FILELIST to erase files. You can also use DISCARD from PEEK and RDRLIST environments. This is discussed under "What You Can Do to the Files in Your Virtual Reader." The FORMAT command erases *all* files on a particular disk.

Action	command(s)	Description
Erase specific files	ERASE	Chapter 1
Erase files from FILELIST menu	DISCARD	Chapter 3
Erase all files on a particular disk	FORMAT	Chapter 1

The *VM/SP CMS Command and Macro Reference* contains information on the following CMS commands that you can use to erase files:

```
DISCARD
ERASE
FORMAT
```

Modifying Files

The System Product editor, invoked with the XEDIT command, allows you to interactively make changes, additions, or deletions to your CMS files. The UPDATE command and the XEDIT command with the UPDATE option provide a way for you to modify a source program without affecting the original.

Action	Command(s)	Description
Update Assembler language programs	UPDATE	Chapter 8
	XEDIT (UPDATE option)	Chapter 8
Edit a file	XEDIT	Chapter 5

The command formats and usage notes for the UPDATE and XEDIT commands are documented in the *VM/SP CMS Command and Macro Reference*.

Moving Files

CMS commands allow you to move a file or copies of file from one place to another; from one virtual disk to another, to or from a tape to a disk, or from your disk to the virtual reader of another user. Some commands, such as the TAPE command, move a copy of the file to another location. Other commands, such as the CP TRANSFER command, move (not copy) files.

Action	Command(s)	Description
Move files from your virtual reader to the reader of another virtual machine	CP TRANSFER	Chapter 7
Dump contents of a virtual disk onto tape, restore such files to disk	DDR	Chapter 6
Move files from tapes to disk, disk to tape	TAPE	Chapter 6
Move an OS partitioned dataset into a CMS file	MOVEFILE	Chapter 9
Move files from your disk to the reader of another (or your own) virtual machine	SENDFILE	Chapter 7
Move files from your virtual reader onto your read/write disk	RECEIVE	Chapter 7

The CP TRANSFER command is documented in the *VM/SP CP Command Reference for General Users*.

The formats and usage notes for the CMS commands

```
DDR
MOVEFILE
RECEIVE
SENDFILE
TAPE
```

are documented in the *VM/SP CMS Command and Macro Reference*.

Developing and Testing CMS Programs

CMS provides commands and macros for assembler language programmers who may need to write programs to be used in the CMS environment.

Action	Command(s) or Macro	Description
Create a module from a program that uses OS or CMS macros	GENMOD	Chapter 8
Identify macro libraries to be used when assembling programs	GLOBAL	Chapter 8
Verify existence of a file	FSSTATE macro	Chapter 8
Identifying file by its FSCB	FSCB macro	Chapter 8
Closing files	FSCLOSE macro	Chapter 8
Make a backup copy	COPYFILE XEDIT and SET FNAME subcommand	Chapter 8 Chapter 8
Modifying source programs	XEDIT with CTL option UPDATE	Chapter 8 Chapter 8
Assemble a program	ASSEMBLE	Chapter 8

The XEDIT SFNAME subcommand is described in the *VM/SP System Product Editor Command and Macro Reference*. The CMS commands and macros:

```
ASSEMBLE
COPYFILE
FSCB
FSCLOSE
FSSTATE
GENMOD
GLOBAL
UPDATE
```

are described in the *VM/SP CMS Command and Macro Reference*.

Developing and Testing OS Programs

CMS simulates many functions of the Operating System (OS), allowing you to create, execute and debug your OS programs interactively.

Action	Command(s)	Description
Identify OS input or output files to CMS	FILEDEF	Chapter 9
Copy OS (or CMS) files from one device to another	MOVEFILE	Chapter 9
Identify Macro libraries to be searched	GLOBAL	Chapter 9
Create, compress, or list macro libraries	MACLIB	Chapter 9
Create CMS files from OS data sets	MOVEFILE	Chapter 9
Assemble assembler language source programs into object module format	ASSEMBLE	Chapter 9
Load relocatable object file into storage	LOAD	Chapter 9
Begin execution of previously loaded program	START	Chapter 9
Read object file and update CMS TEXT libraries	TXTLIB	Chapter 9
Create LOADLIB libraries from OS object modules	LKED	Chapter 9

The command formats and usage notes for the following CMS commands:

```
ASSEMBLE
FILEDEF
GLOBAL
LKED
LOAD
MACLIB
MOVEFILE
START
TXTLIB
```

are documented in the *VM/SP CMS command and Macro Reference.*

Developing and testing VSE Programs

You can use CMS to create and compile VSE programs. CMS simulates many functions of VSE so that you can use VM/SP to develop your programs and then execute them in a VSE virtual machine.

Action	Command(s)	Description
Entering CMS/DOS environment	SET DOS ON	Chapter 10
Accessing the system residence volume	ACCESS	Chapter 10
Display fileids of files on a DOS disk	LISTDS	Chapter 10
Create CMS files from DOS files on tape	MOVEFILE and FILEDEF	Chapter 10
Create CMS files for VSE modules from tape	VMFDOS	Chapter 10
Assign logical units	ASSGN	Chapter 10
Supply CMS/DOS with specific file identification information for a file that is going to be used for input or output	DLBL	Chapter 10
Copy, punch, display at terminal, or print books from private or system source statement libraries	SSERV	Chapter 10
Copy, punch, display at terminal, or print relocatable modules	RSERV	Chapter 10
Copy, punch, display at terminal, or print procedures from system procedure library	PSERV	Chapter 10
Copy and de-edit macros from system and private E sublibraries	ESERV	Chapter 10
Access the directories of system or private libraries	DSERV	Chapter 10
Link-edit relocatable modules	DOSLKED	Chapter 10
Load phases from either system or private DOS core image libraries	FETCH	Chapter 10
Identify macro libraries to be searched	GLOBAL	Chapter 10

The VMFDOS command is described in the *VM/SP Installation Guide.*

The following CMS commands:

```
ACCESS
ASSGN
DLBL
DSERV
DOSLKED
```

```
ESERV
FETCH
FILEDEF
GLOBAL
LISTDS
MOVEFILE
PSERV
RSERV
SET
SSERV
```

are described in the *VM/SP CMS Command and Macro Reference*.

What You Can Do to Your VSAM Catalogs

You can use CMS commands to obtain information about your VSAM catalogs.

Action	Command(s)	Description
Verify a complete catalog structure	CATCHECK	Chapter 11
Determine what free space is available for allocation	LISTDS	Chapter 11
Define a ddname and relate it to a disk file	DLBL	Chapter 11
Define and maintain VSAM catalogs and data sets	AMSERV	Chapter 11
Erase LISTING files from your disk	DISCARD (from FILELIST) ERASE	Chapter 3 Chapter 3
Using tapes with VSAM	TAPE	Chapter 19
Obtain VSE/VSAM assembler language macros	VSEVSAM	Chapter 19

The VSEVSAM command is documented in the *VM/SP Installation Guide*. The following CMS commands:

```
AMSERV
CATCHECK
DISCARD
DLBL
ERASE
LISTDS
TAPE
```

are documented in the *VM/SP CMS Command and Macro Reference*:

Interactive Debugging

If you encounter problems, executing application programs or when you want to test new lines of code, you can use a variety of CMS and CP debugging techniques to examine your program as it executes.

Action	Command(s)	Description
Enter debug environment	DEBUG	Chapter 2
	CP EXTERNAL	Chapter 2
Stop execution at a particular virtual address	CP PER	Chapter 13
	BREAK subcommand	Chapter 13
	CP ADSTOP	Chapter 13
Examine virtual storage, registers, or PSW	CP DISPLAY	Chapter 13
Change the contents of a storage location, register, or control word.	CP STORE	Chapter 13
Resume program execution	CP BEGIN	Chapter 13
Trace program activity	CP PER	Chapter 13
	SET EXECTRAC	Chapter 18
	SVCTRACE	Chapter 13
	CP TRACE	Chapter 13
Obtain a program dump	CP DUMP	Chapter 13
Stop all tracing of your System Product interpreter EXEC	TE Immediate command	Chapter 18
Start tracing of your System Product interpreter EXEC	TS Immediate command	Chapter 18

Information about the following CP commands is found in the *VM/SP CP Command Reference for General Users.*

```
ADSTOP
BEGIN
DISPLAY
DUMP
EXTERNAL
PER
STORE
TRACE
```

Information about the following:

```
DEBUG subcommands that you can enter in the debug
environment
SVCTRACE command
SET EXECTRAC command
TE Immediate command
TS Immediate command
```

is found in the *VM/SP CMS Command and Macro Reference.*

Chapter 5. Editing Your Files

To edit a file means to make changes, additions, or deletions to a CMS file that is on a disk, and to make these changes interactively: you instruct the editor to make a change, the editor does it, and then you request another change. You can edit a file that does not exist; when you do so, you create the file online, and can modify it as you enter it.

To file a file means to write a file you are editing back onto a disk, incorporating any changes you made during the editing session. When you issue the FILE subcommand to write a file, you are no longer in edit environment, but are returned to the CMS environment. You can, however, write a file to disk and then continue editing it, by using the SAVE subcommand.

An editing session is the period of time during which a file is in your virtual storage area, from the moment you issue the XEDIT command or the EDIT command until you let the editor know that you are finished working on the file, by entering FILE or QUIT.

Editors Available for You to Use

In CMS usage, the term edit is used in a variety of ways, all of which refer, ultimately, to the functions of the System Product editor or the CMS editor.

The System Product Editor

The System Product editor provides full screen and file manipulation capabilities not offered by the CMS editor.

This editor has the following advantages:

1. Full screen support for IBM 3270 Display Terminals is available including:
   - the ability to display multiple views of the same file or of different files.
   - automatic "wrapping" of lines that are wider than a screen line
   - the ability to enter selected (prefix) subcommands directly on the displayed lines.
   - the ability to define the screen format according to individual preferences.
2. Extended string search facilities are provided for improved text processing.
3. A variety of macros, that use the EXEC 2 interpreter are offered.
4. An enhanced set of functions to handle program development is available, including automatic update generation.
5. The ability to import and export data between files is provided.

For complete information about the System Product editor, see the *VM/SP System Product Editor User's Guide* and the *VM/SP System Product Editor Command and Macro Reference.*

The CMS Editor

When you issue the EDIT command, the System Product editor automatically places you in CMS editor (EDIT) migration mode. In this mode, you can issue both EDIT and XEDIT subcommands. For complete information on EDIT compatibility mode, see the *VM/SP System Product Editor Command and Macro Reference*. For more information on using the CMS editor, see Appendix A, "The CMS Editor."

The XEDIT Command

When you issue the XEDIT command you must specify the filename and filetype of the file you want to edit. For Example:

```
XEDIT NEWFILE SCRIPT
```

Writing a File Onto Disk

A file you create and the modifications that you make to it during an edit session are not automatically written to a disk file. To save the results, you can do the following:

- Periodically issue the subcommand:

  ```
  save
  ```

  to write onto disk the contents of the file as it exists when you issue the subcommand. Periodically issuing this XEDIT subcommand protects your data against a system failure; you can be sure that changes you make are not lost.

- At the beginning of the edit session, issue the SET AUTOSAVE subcommand, with a number:

  ```
  set autosave 10
  ```

  Then, for every tenth change or addition to the file, the editor issues an automatic save request, which writes the file onto disk.

- To terminate the editing session and write the file onto disk, issue the subcommand:

  ```
  file
  ```

  The file disappears from your screen, but the editor saved it on your disk. You can return to the edit environment by issuing the XEDIT command, specifying a different file or the same file.

  The editor decides which disk to write the file onto according to the following hierarchy:

- If you specify a filemode on the FILE or SAVE subcommand line, the file is written onto the specified disk.

- If the current filemode of the file is the mode of a read/write disk, the file is written onto that disk. (If you have not specified a filemode letter, it defaults to your A-disk.)

- If the filemode is the mode of a read-only extension of a read/write disk, the file is written onto the read/write parent disk.
- If the filemode is the mode of a read-only disk that is not an extension of a read/write disk, the editor cannot write the file and issues an error message.

If you are editing a file and decide that you do not wish to save the changes, you can use the subcommand:

```
quit
```

No changes that you made since you last used the SAVE subcommand (or the editor last issued an automatic save for you) are retained. If you have just begun an edit session, and have made no changes at all to a file, and for some reason you do not want to edit it at all (for example, you misspelled the name, or want to change a CMS setting before editing the file), you can use the QUIT subcommand instead of the FILE subcommand to terminate the edit session and return to CMS.

A file must have at least one line of data in order to be written. To create a new file called SHOPPING LIST, enter:

```
xedit shopping list
```

The XEDIT command invokes the System Product editor, so you will see looks like Figure 5-1.

```
SHOPPING LIST     A1  F 80  TRUNC=80 SIZE=0 LINE=0 COL=1 ALT=0




===== * * * TOP OF FILE * * *
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== * * * END OF FILE * * *




====> input_
                                                        X E D I T  1 FILE
```

Figure 5-1. Sample XEDIT Screen

On the command line (next to the arrow) type INPUT and press the ENTER key. The file is placed in input mode. The cursor is placed automatically on the first line in the input zone, where you can enter your data. You are writing input lines that are eventually going to be written onto your A-disk.

Enter the following data:

```
apples
lettuce
tomatoes
bread
```

```
 SHOPPING LIST      A1  F 80   TRUNC=80  SIZE=0  LINE=0 COL=1 ALT=3
INPUT MODE:




* * * TOP OF FILE * * *
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....
apples
lettuce
tomatoes
bread



====> * * * INPUT ZONE * * *
                                                    INPUT-MODE 1 FILE
```

Figure 5-2. Sample XEDIT Screen In INPUT Mode

Now, press the ENTER key, the screen moves up so that you can enter more data. When you are finished entering data, press the ENTER key again to return to edit mode.

To keep this file in permanent storage, you type FILE on the command line and press the ENTER key. You should see a message that looks something like this:

```
R;
```

Even though the file has disappeared from your screen, the editor has saved it on your disk.

Let's check and see if the file was really saved. We'll use the FILELIST command to list the files on your A-disk with a filename of shopping. Enter:

```
filelist shopping
```

The display may look like Figure 5-3.

```
  MYLOGON FILELIST    A0  V 108  TRUNC=108 SIZE=1    LINE=1 COL=1 ALT=0
Cmd Filename Filetype Fm Format Lrecl Records Blocks   Date    Time
_   SHOPPING LIST     A1 F         80       4      1  5/16/83 15:07:49




1= Help     2= Refresh  3= Quit    4= Sort(type) 5= Sort(date) 6= Sort(size)
7= Backward 8= Forward  9= FL /n  10=           11= XEDIT     12= Cursor

====>
                                                                  XEDIT
```

Figure 5-3. Sample FILELIST Screen for a Particular Filetype

Using the Editor in Line Mode

The editor's display mode is the most common format of operation on a 3270. There are, however, instances when it is not possible or not desirable to use the editor in display mode. For these instances, you should use the line mode of operation, which is the equivalent to using a typewriter terminal. When you use line mode, each XEDIT subcommand you enter, and the response (if you have verification on), is displayed, a line at a time, on the screen in the output display area. There is no full screen display of the file.

You need only be concerned with using line mode if you are connected to VM/SP by a remote 3270 line, or if you are editing a file from within an EXEC and you want to control the screen display. Although it is possible to use the editor in line mode on a local 3270, it is rarely necessary for normal editing purposes.

Editing on a Remote 3270

When you invoke the editor from a remote 3270, you are placed in line mode by the editor. The advantage of using the 3270 in line mode (particularly on a remote editor) is that the editor can respond more quickly to display requests. When you use display mode, the editor has to write out the entire output display area when you move the current line pointer; in line mode, it has only to write a single line.

If you want to use display mode, you enter the XEDIT subcommand:

```
set terminal display
```

The editor begins operating in display mode, and you can use the special editing functions available in display mode.

However, when you are using a remote 3270 in display mode, and you enter the INPUT subcommand to begin entering input lines, the screen is cleared, and your input lines are displayed as if you were in line mode, beginning at the top of the screen. When you enter a null line to return to edit mode, the editor returns to a full screen display.

You can resume editing in line mode by using the subcommand:

```
set terminal typewriter
```

Editing From an EXEC File

If you invoke the editor from an EXEC, but you do not want the screen cleared when the editor gets control, you can specify the NOCLEAR option on the XEDIT command line:

```
xedit test file (noclear
```

This places the 3270 in line mode, so that the lines already on the screen are not erased.

The 3270 remains in line mode for the remainder of the edit session, and you cannot use the SET TERMINAL subcommand to place it in display mode.

Chapter 6. Using Real Printers, Punches, Readers, and Tapes

CMS Unit Record Device Support

CMS supports one virtual reader at address 00C, one virtual punch at address 00D, and one virtual printer at address 00E. When you invoke a CMS command or execute a program that uses one of these unit record devices, the device must be attached at the virtual address indicated.

Using the CP Spooling System

Any output that you direct to your virtual printer or punch, or any input you receive from your reader, is controlled by the spooling facilities of the control program (CP). Each output unit is known to CP as a spool file, and is queued for processing with the spool files of other users on the system. Ultimately, a spooled printer file or a spooled punch file may be released to a real printer or card punch for printing or punching.

The final disposition of a unit record spool file depends on the spooling characteristics of your virtual unit record devices, which you can alter with the CP command SPOOL. To find out the current characteristics of your unit record devices you can issue the command:

```
cp query ur
```

Figure 6-1 is an example of the response you will receive from issuing this command.

```
RDR 00C CL A  NOCONT   HOLD   EOF        READY
PUN 00D CL A  NOCONT NOHOLD COPY 001     READY FORM STANDARD
    00D TO  CMSGDE   DIST 2G47-706
PRT 00E CL A  NOCONT NOHOLD COPY 001     READY FORM STANDARD
    00E TO  CMSGDE   DIST 2G47-706  FLASHC 000
    00E FLASH       CHAR       MDFY        FCB
```

Figure 6-1. CP QUERY Unit Record Response

You can use the SPOOL command to change spool file characteristics. When you use the SPOOL command to control a virtual unit record device, you do not change the status of spool files that already exist, but rather set the characteristics for subsequent output. For information on modifying existing spool files, see "Altering Spool Files," below.

Spool File Characteristics

> **Note:** When you issue a SPOOL command for a unit record device, you can refer to it by its virtual address, as well as by its generic device type (for example, CP SPOOL E HOLD).

CLASS (CL):

Spool files, in the CP spool file queue, are grouped according to class, and all files of a particular class may be processed together, or directed to the same real output device. The default values for your virtual machine are set in your VM/SP directory entry, and are probably the standard classes for your installation.

You may need, however, to change the class of a device if you want a particular type of output, or some special handling for a spool file. For example, if you are printing an output file that requires special forms, and your installation expects that output to be spooled class Y, issue the command:

```
cp spool printer class y
```

All subsequent printed output directed to your printer at virtual address 00E (all CMS output) is processed as class Y.

HOLD:

If you place a HOLD on your printer or punch, any files that you print or punch are not released to the control program's spooling queue until you specifically alter the hold status. By placing your output spool files in a hold status, you can select which files you print or punch, and you can purge duplicate or unwanted files. To place printer and punch output files in a hold status issue the commands:

```
cp spool printer hold
cp spool punch hold
```

When you have placed a hold status on printer or punch files and you produce an output file for one of these devices, CP sends you a message to remind you that you have placed the file in a hold:

```
PRT FILE xxxx FOR userid COPY xx HOLD
```

If, however, you have issued the command:

```
cp set msg off
```

then you do not receive the message.

When you place a reader file in a hold status, then the file remains in the reader until you remove the hold status and read it, or you purge it.

COPY:

If you want multiple copies of a spool file, you should use the COPY operand of the SPOOL command:

```
cp spool printer copy 10
```

If you enter this command, then all subsequent printer files that you produce are each printed 10 times, until you change the COPY attribute of your printer.

FOR:

You can spool printed or punched output so that it will be distributed to another userid by using the FOR operand of the SPOOL command. For example, if you enter:

```
cp spool printer for charlie
```

Then, all subsequent printer files that you produce have, on the output separator page, the userid CHARLIE and the distribution code for that user. The spool file is then under the control of that user, and you cannot alter it further.

CONT, NOCONT:

You can print or punch separate spool files with the NOCONT option of the CP SPOOL command. You can also combine them into one continuous spool file if you use the CONT operand of the CP SPOOL command. For example, if you issue the following sequence of commands:

```
cp spool punch cont to brown
punch asm1 assemble
punch asm2 assemble
punch asm3 assemble
cp spool punch nocont
cp close punch
```

Then, the three files ASM1 ASSEMBLE, ASM2 ASSEMBLE, and ASM3 ASSEMBLE, are punched to user BROWN as a single spool file. When user BROWN reads this file onto a disk, however, CMS creates separate disk files. Note that if multiple files are sent with continuous spooling (using CP SPOOL PUNCH CONT) and a series of DISK DUMP commands, RECEIVE recognizes only the first file identifier (filename and filetype). Any files having the same file identifier as existing files on your A-disk will overlay those files on your A-disk.

As a sender, you can avoid imposing this problem on file recipients by doing any of the following:

1. Always use SENDFILE, which resets any continuous spooling options in effect.

2. Do not spool the punch continuous.

3. If you must send files with continuous spooling, warn the recipient(s) that files are being sent in this manner and list the file identifiers of the files you are sending.

Similarly, if the punch is spooled continuous and PUNCH is used to send multiple files, the file is read in as one file with ":READ" cards imbedded. In this case, although no files are overlaid, the recipient must divide the file into individual files. This problem can also be avoided by using SENDFILE or by not spooling the punch continuous.

TO:

When you spool your printer or punch to another userid, all output from that device is transferred to the virtual reader of the userid you specify. When you are punching a CMS disk file, as in the example above, you should use the TO operand of the SPOOL command to specify the destination of the punch file.

You can also use this operand to place output in your own virtual reader by using the * operand:

```
cp spool printer to *
```

After you enter this command, subsequent printed output is placed in your virtual reader. You might use this technique as an alternative way of preventing a printer file from printing, or, if you choose to read the file onto disk from your reader, of creating a disk file from printer output.

Similarly, if you are creating punched output in a program and you want to examine the output during testing, you could enter:

```
cp spool punch to *
```

so that you do not punch any real cards or transfer a virtual punch file to another user.

Altering Spool Files

After you have requested that VM/SP print or punch a file, or after you have received a file in your virtual reader and before the file is actually printed, punched, or read, you can alter some of its characteristics, change its destination, or delete it altogether.

Every spool file in the VM/SP system has a unique four-digit number from 1 to 9900 assigned to it, called a spoolid. You can use the spoolid of a file to identify it when you want to do something to it. You can also change a group of files, by specifying that all files of a particular class be altered in some way, or you can manipulate all of your spool files for a certain device at the same time.

The CP commands that allow you to manipulate spool files are CHANGE, ORDER, PURGE, and TRANSFER. In addition, you can use the CP QUERY command to list the status and characteristics of spool files associated with your userid.

When you use any of these commands to reference spool files of a particular device, you have the choice of referring to the files by class or by spoolid. You can also specify ALL. For example, if you enter the command:

```
cp query printer all
```

you might see the display:

```
ORIGINID FILE CLASS RECORDS CPY HOLD DATE  TIME     NAME     TYPE    DIST
CMSUG    0142 K PRT 000178  002 USER 04/17 07:58:48 SCHED    SCRIPT  BIN706
CMSUG    0180 1 PRT 002021  001 NONE 04/17 08:02:26 TESTFILE SCRIPT  BIN706
```

Until any of these files are processed, or in the case of files in the hold status, until they are released, you can change the spool file name and spool file type (this information appears on the first page or first card of output), the distribution code, the number of copies, the class, or the hold status, using the CP CHANGE command. For example:

```
cp change printer all nohold
```

changes all printer files that are in a hold status to a nohold status. The CP CHANGE command can also change the spooling class, distribution code, and so on.

If you decide that you do not want to print a particular printer file, you can delete it with the CP PURGE command:

```
cp purge printer 7615
```

After you have punched a file to some other user, you cannot change its characteristics or delete it unless you restore it to your own virtual reader. You can do this with the TRANSFER command:

```
cp transfer all from usera
```

This command returns to your virtual reader all punch files that you spooled to USERA's virtual reader.

You can determine, for your reader or printer files, in what order they should be read or printed. If you issue the command:

```
cp order printer 8195 6547
```

Then, the file with spoolid of 8195 is printed before the file with a spoolid of 6547.

The CP spooling system is very flexible, and can be a useful tool, if you understand and use it properly. The *VM/SP CP Command Reference for General Users* contains complete format and operand descriptions for the CP commands you can use to modify spool files.

Using Your Card Punch and Card Reader in CMS

The CMS READCARD command reads cards from your virtual reader at address 00C. Cards can be placed in the reader in one of three ways:

- By reading real punched cards into the system card reader. A CP ID card tells the CP spooling system which virtual reader is to receive the card images.
- By transferring a file from another virtual machine. Cards are transferred as a result of a virtual punch or printer being spooled with the TO operand, or as a result of the TRANSFER command. Virtual card images are created with the CMS PUNCH command, or from user programs or EXEC procedures.
- By punching, spooling, or transferring files to your own reader, or by using the MOVEFILE command.

Using Real Cards

If you have a deck of punched cards that you want read into your virtual machine reader, you should punch, preceding the deck, a CP ID card. If your userid is HAPPY, then the id card would be:

```
ID HAPPY
```

If you plan to use the READCARD command to read this file onto a CMS disk, you can also punch a READ control card that specifies the filename and filetype you want to have assigned to the file:

```
:READ PROG6 ASSEMBLE
```

Then, to read this file onto your CMS A-disk, you can enter the command:

```
readcard *
```

If a file named PROG6 ASSEMBLE already exists, it is replaced.

If you do not punch a READ control card, you can specify a filename and filetype on the READCARD command:

```
readcard prog6 assemble
```

If this spool file contained a READ control card, the card is not read, but remains in the file; if you edit the file, you can use the DELETE subcommand to delete it.

If a file does not have a READ control card, and if you do not specify a filename and filetype when you read the file, CMS names the file READCARD CMSUT1.

If you are reading many files into the real system card reader, and you want to read them in as separate spool files (or you want to spool them to different userids), you must separate the cards and read the decks onto disk individually. The CP system, after reading an ID card, continues reading until it reaches a physical end of file.

Using Your Virtual Card Punch

When you use the CMS PUNCH command to punch a spool file, a READ control card is punched to precede the deck, so that it can be read with the READCARD command. If you do not wish to punch a READ control card (also referred to as a header card), you can use the NOHEADER option on the PUNCH command:

```
punch prog8 assemble * ( noheader
```

You should use the NOHEADER option whenever you punch a file that is not going to be read by the READCARD command.

The PUNCH command can only punch records of up to 80 characters in length. If you need to punch or to transfer to another user a file that has records greater than 80 characters in length, you can use the DISK DUMP command:

```
disk dump prog9 data
```

If your virtual punch has been spooled to another user, that user can read this file using the DISK LOAD command:

```
disk load
```

Unlike the READCARD command, DISK LOAD does not allow you to specify a file identification for a file you are reading; the filename and filetype are always the same as those specified by the DISK DUMP command that created the spool file.

A card file created by the DISK DUMP command can only be read onto disk by the DISK LOAD command.

Using the MOVEFILE Command

You can use the MOVEFILE command, in conjunction with the FILEDEF command, to place a file in your virtual reader, or to copy a file from your reader to another device. For example:

```
cp spool punch to *
filedef output punch
filedef input disk coffee exec a1
movefile input output
```

the file COFFEE EXEC A1 is punched to your virtual card punch (in card-image format) and spooled to your own virtual reader.

Creating Files Using Your Punch

Apart from the procedures shown above that transfer whole files with one or two commands, there are other methods you can use to create files using your virtual punch. From a program or an EXEC file, you can punch one line at a time to your virtual punch. Then use the CLOSE command to close the spool file:

```
cp close punch
```

Depending on how the punch was spooled (the TO setting), the virtual punch file is either punched or transferred to a virtual reader.

Punching Cards Using I/O Macros:

If you write an OS, DOS, or CMS program that produces punched card output, you should make an appropriate file definition. If you are an OS user, you should use the FILEDEF command to define the punch as an output data device; if you are a DOS user, you must use the ASSGN command. If you are using the CMS PUNCHC macro, the punch is assigned for you. The spooling characteristics of your virtual punch control the destination of the punched output.

Punching Cards From an EXEC:

The CMS EXEC facility provides two control statements for punching cards: &PUNCH, which punches a single line to the virtual punch, and &BEGPUNCH, which precedes a number of lines to be punched. In a System Product interpreter, EXEC 2, or CMS EXEC, you can use the CMS commands PUNCH and DISK DUMP to punch CMS files.

Handling Tape Files in CMS

There are a variety of tape functions that you can perform in CMS, and a number of commands that you can use to control tape operations or to read or write tape files. One of the advantages of placing files on tapes is portability: it is a convenient method of transferring data from one real computing system to another. In CMS, you can use tapes created under other operating systems. There are also two CMS commands, TAPE and DDR, that create tape files in formats unique to CMS, that you can use to back up minidisks or to archive or transfer CMS files.

Under VM/SP, virtual addresses 181 through 184 are usually reserved for tape devices. In most cases, you can refer to these tapes in CMS by using the symbolic names TAP1 through TAP4. In any event, before you can use a tape, you must have it mounted and attached to your virtual machine by the system operator. When the tape is attached, you receive a message. For example, if the operator attaches a tape to your virtual machine at virtual address 181, you receive the message:

```
TAPE 181 ATTACHED
```

The various types of tape files, and the commands and programs you can use to read or write them are:

TAPE Command

The CMS TAPE command creates tape files from CMS disk files. They are in a special format, and should only be read by the CMS TAPE LOAD command. For examples of TAPE command operands and options, see "Using the CMS TAPE Command."

TAPPDS Command:

The TAPPDS command creates CMS disk files from OS or DOS sequential tape files, or from OS partitioned data sets.

TAPEMAC Command:

The TAPEMAC command creates CMS MACLIB files from OS macro libraries that were unloaded onto tape with the IEHMOVE utility program.

MOVEFILE Command:

The MOVEFILE command can copy a sequential tape file onto disk or a disk file onto tape. It can move files from your reader to tape or from tape to your punch.

User Programs:

You can write programs that read or write sequential tape files using OS, DOS, or CMS macros.

Access Method Services:

Tapes created by the EXPORT function of access method services can be read only using the access method services IMPORT function. Both the IMPORT and EXPORT functions can be invoked in CMS using the AMSERV command. The access method services REPRO function can also be used to copy sequential tape files.

DDR Program:

The DDR program, invoked with the CMS command DDR, dumps the contents of a virtual disk onto tape, and should be used to restore such files to disk.

Using the CMS TAPE Command

The CMS TAPE command provides a variety of tape handling functions. It allows you to selectively dump or load CMS files to and from tapes, as well as to position, rewind, and scan the contents of tapes. You can use the TAPE command to save the contents of CMS disk files, or to transfer them from one VM/SP system to another. The following example shows how to create a CMS tape with three tape files on it, each containing one or more CMS files, and then shows how you, or another user, might use the tape at a later time.

The example is in the form of a terminal session and shows, in the "Terminal Display" column, the commands and responses you might see. System messages and responses are in uppercase, and user-entered commands are in lowercase. The "Comments" column provides explanations of the commands and responses.

Terminal Display	Comments
`TAPE 181 ATTACHED`	Message indicates that the tape is attached.

`listfile * assemble a (exec`
`R;`
`cms tape dump`
`TAPE DUMP PROG1 ASSEMBLE A1`
	Prepare to dump all ASSEMBLE files by using the LISTFILE command EXEC option; then execute the CMS EXEC using TAPE and DUMP as arguments.

`DUMPING.....`
`PROG1 ASSEMBLE A1`
`TAPE DUMP PROG2 ASSEMBLE A1`
`DUMPING.....`
`PROG2 ASSEMBLE A1`
`TAPE DUMP PROG3 ASSEMBLE A1`
`.`
`.`
`.`
	The TAPE command responds to each TAPE DUMP by printing the file identification of the file being dumped.

`TAPE DUMP PROG9 ASSEMBLE A1`
`DUMPING.....`
`PROG9 ASSEMBLE A1`
`R;`
	The last file, PROG9 ASSEMBLE, is dumped.

`tape wtm` `R;`	TAPE command writes a tape mark to indicate an end of file.

`tape dump mylib maclib a`
`DUMPING.....`
`MYLIB MACLIB A1`
`R;`
`tape dump cmslib maclib *`
`DUMPING.....`
`CMSLIB MACLIB S2`
`R;`
	Two macro libraries are dumped, by specifying the file identifiers.

`tape wtm` `R;`	Another tape mark is written.

`tape dump mylib txtlib a`
`DUMPING.....`
`MYLIB TXTLIB A1`
`R;`
	A TEXT library is dumped.

`tape wtm 2` `R;`	Two tape marks are written to indicate the end of the tape.
`tape rew` `R;`	The tape is rewound.

`tape scan (eof 4`
`SCANNING....`
`PROG1 ASSEMBLE A1`
`PROG2 ASSEMBLE A1`
`PROG3 ASSEMBLE A1`
`PROG4 ASSEMBLE A1`
`PROG5 ASSEMBLE A1`
`PROG6 ASSEMBLE A1`
`PROG7 ASSEMBLE A1`
`PROG8 ASSEMBLE A1`
`PROG9 ASSEMBLE A1`
	The tape is scanned to verify that all of the files are on it.

Terminal Display	Comments

`END-OF-FILE OR END-OF-TAPE`
`MYLIB MACLIB A1`
`CMSLIB MACLIB S2`
`END-OF-FILE OR END-OF-TAPE`
`MYLIB TXTLIB A1`
	Tape mark indication.

`END-OF-FILE OR END-OF-TAPE`
`END-OF-FILE OR END-OF-TAPE`
`R;`
	Two tape marks indicate the end of the tape.

`#cp det 181` `TAPE 181 DETACHED`	The CP DETACH command rewinds and detaches the tape.
******* The tape created above is going to be read.*******	
`TAPE 181 ATTACHED`	Message indicating the tape is attached.
`tape load prog4 assemble`	One file is to be read onto disk.

`LOADING.....`
`PROG4 ASSEMBLE A1`
`R;`
	The TAPE command displays the name of the file loaded. Any existing file with the same filename and filetype is erased.

`tape scan`
`SCANNING....`
`PROG5 ASSEMBLE A1`
`PROG6 ASSEMBLE A1`
`PROG7 ASSEMBLE A1`
`PROG8 ASSEMBLE A1`
	The remainder of the first tape file is scanned.

`END-OF-FILE OR END-OF-TAPE` `R;`	Indication of end of first tape file.

`tape scan`
`SCANNING....`
`MYLIB MACLIB A1`
`CMSLIB MACLIB S2`
`END-OF-FILE OR END-OF-TAPE`
`R;`
	The second tape file is scanned.

`tape bsf 2` `R;`	The tape is backed up and positioned in front of the last tape file.
`tape fsf` `R;`	The tape is forward spaced past the tape mark.

`tape load (eof 2`
`LOADING.....`
`MYLIB MACLIB A1`
`CMSLIB MACLIB A2`
`END-OF-FILE OR END-OF-TAPE`
`MYLIB TXTLIB A1`
`END-OF-FILE OR END-OF-TAPE`
`R;`
	The next two tape files are going to be read.

`#cp detach 181` `TAPE 181 DETACHED`	The tape is detached.

Tape Labels in CMS

Support in the CMS component of VM/SP to process labelled tapes includes the following features:

- Checks IBM standard labels on input
- Writes IBM standard labels on output
- Allows you to specify routines to process standard user labels during DOS and OS macro simulation under CMS
- Allows you to specify exits for processing tapes with nonstandard labels during execution of CMS macro simulations and some CMS tape operation commands CMS processes all tape labels; CP does not process tape labels.

Limitations

CMS tape label processing does not include:

- Label processing for tapes that are read backwards
- Processing of multivolume files on tapes
- Support for ANSI tapes or ASCII labels
- Label processing for any functions of the CMS TAPE command except the two functions DVOL1 and WVOL1 that process VOL1 labels.

User Responsibilities

You must initiate all your own tape label processing. To specify that you have a labelled tape, use the FILEDEF command for an OS simulation program, or use a DOS DTFMT macro for a CMS/DOS program. You can also use the TAPESL macro to process standard HDR1 and EOF1 labels and the CMS TAPE command to write and display standard VOL1 labels. You can provide IBM standard label description details with the LABELDEF command for all types of label processing. After label processing has been requested, it occurs automatically and there is no interaction between you and CMS unless an error occurs. See the "Error Processing" section later in this publication for a discussion of error processing.

Label Processing in OS Simulation

If you are running an OS simulation program and using OPEN and CLOSE macros, you specify the type of label processing you want in a FILEDEF command for a given file. Detailed information about the FILEDEF command is found in the *VM/SP CMS Command and Macro Reference*. You may specify that you want standard label processing (with SL) or nonstandard label processing (with NSL). If you choose nonstandard label processing, you must already have written a routine to process nonstandard labels. The name of this routine must be specified by the filename in the NSL parameter on FILEDEF. An example of nonstandard label processing is given in the section "NSL Processing." To be sure that the tape you are using contains no IBM labels, you may specify no label processing (NL) in the FILEDEF command. When NL is specified, CMS does not open files on a tape containing a VOL1 label as its first record. You also can specify bypass tape label processing (BLP) on a FILEDEF command. BLP tells CMS to bypass tape label processing for a file, and instead, to position the tape at a particular file before

processing the data records in the file. If you specify LABOFF for a FILEDEF tape file, label processing is turned off and there is no tape positioning or label checking.

LABOFF is the default, so you do not receive any processing or tape positioning for a tape file unless you specifically request it. If you specify BLP, NL, SL, or SUL processing but omit a positional parameter, the position defaults to 1 and the tape is positioned at the first file. Examples of NL, BLP, and LABOFF processing are given in the sections "No Label (NL) Processing," "Bypass Label (BLP) Processing," and "Label Off (LABOFF) Processing."

IBM Standard Tape Label Processing

For IBM standard labels, you specify, SL or SUL, and optional positional and VOLID parameters. On a FILEDEF command, SUL means standard user labels. Everything you do for SL files, you must also do for SUL files. The positional parameter for standard label files works the same way it does in OS/VS. If you specify:

```
filedef filex tap1 sl 2
```

the tape is spaced to what is physically the fourth file on the tape before processing begins. The reason for this spacing is that a standard labelled tape has one header file, one data file, and one trailer file for each data file. If you leave off the positional parameter:

```
filedef filey tap3 sul
```

you get the first file on the tape.

The optional VOLID parameter on the FILEDEF command allows you to specify the volume serial number in the VOL1 label of a tape in case you want only the VOL1 label checked on the tape. If you want to specify other fields in IBM standard labels, you must also provide a LABELDEF statement for the tape file. The LABELDEF statement allows you to assign values to all fields in a standard HDR1 or EOF1 label. A complete description of how the LABELDEF command works may be found in the "LABELDEF Command" section later in this publication.

The following command defines filez as a standard labelled tape file on a tape with a VOL1 label and a volume serial number of DEPT78:

```
filedef filez tap1 sl volid dept78
```

If you also wish to specify a data set identifier for filez, you must furnish a LABELDEF command for filez as well as the FILEDEF command. Data set name may not be specified on the FILEDEF command. The LABELDEF statement below assigns a data set name of payroll to filez.

```
labeldef filez fid payroll
```

You can also specify file sequence number, volume sequence number, expiration date and other fields on a LABELDEF command. However, if you are using OS simulation macros (OPEN, CLOSE, READ, WRITE, GET, PUT, etc.) to process your tape file, the only LABELDEF parameter that has meaning for input files is fid (data set identifier). This is the only field that is checked on input by OS simulation. The other LABELDEF fields are used to specify values to be written in output labels. They are also used by other types of tape label processing

(CMS/DOS and CMS) to check input labels. If no LABELDEF command has been supplied for output files, default values are used to write out labels (see the section on the LABELDEF command for the default values).

After you have set up your descriptive information for a standard labelled tape file in FILEDEF and LABELDEF statements, you run a regular OS simulation program under CMS. During program execution, HDR1 and HDR2 labels are written or checked at OPEN time. EOF1 and EOF2 labels are written or checked at CLOSE time. To have EOF labels processed, you must issue a CLOSE macro. The VOL1 label on a tape is checked whenever a file on that tape is opened if the user has specified a VOLID parameter on his FILEDEF statement or LABELDEF statement for the file. If the volid is specified on both LABELDEF and FILEDEF, the more recent specification is used. If no volid is specified, it is not checked. After checking the volid, the tape is positioned and the HDR label is processed. For processing multifile volumes, you may wish to use the LEAVE option on the FILEDEF command. This option prevents a tape from being rewound and positioned before each tape file is processed. The LEAVE option does not exist on an OS DD statement.

For input files, HDR2 and EOF2 labels are skipped. There is no merge of information from a HDR2 label with information in the DCB as there is under an OS/VS operating system. Output HDR2/EOF2 records are written from information in the DCB and the CMSCB (FCBSECT). Note that the tape density and TRTCH fields in HDR2/EOF2 records are taken from what the user specifies in his FILEDEF command for the tape file. They may not correspond to the actual density and TRTCH fields used to write the tape.

To process standard user labels in OS simulation, you must do the following:

1. Specify the file as SUL in a FILEDEF command.

2. Provide a routine to process the user standard labels in your program.

3. Put the address of the user label routine in the DCB EXIT list of the DCB for the file. See the IBM publication *OS/VS1 Data Management Services Guide* or *OS/VS2 MVS Data Management Services Guide*, for instructions on how to establish a DCB EXIT list, and the exact linkage for communication between user label routines and the operating system. This exact linkage should be used under CMS with the following exceptions:

   a. There is no support for code x'06' EOV EXIT routine.

   b. For input labels, return codes 8 and 12 from the user routine are not supported. If an input return code is not 0, it is treated as if it were 4.

4. Note that your standard user label routines do not perform any input/output. They set up an output label for writing, but the CMS tape label processing routines actually write out the label. For input, the CMS label processing routines read in your user standard label but then give control to your routine to check the label.

No Label (NL) Processing

You should specify NL in the FILEDEF command when you expect a tape does not contain any IBM standard tape labels. CMS reads your tape at the time a file

is opened and does not open the file if the tape contains a VOL1 label as its first record. If the tape does not contain a VOL1 label, a file is opened and the tape is positioned by using the position parameter (n). For example, if you specify:

```
filedef fileq tap1 nl 2
```

fileq is not opened if the tape on tap1 (181) has a VOL1 label. If the tape does not have a VOL1 label, fileq is opened and the tape is positioned at the second file. If you do not specify a position parameter, the tape is positioned at the first file, (that is, the load point).

Bypass Label (BLP) Processing

You should specify BLP in the FILEDEF command to bypass tape label processing. CMS does not check your tape for an IBM standard tape label. It uses the position parameter you specified to position the tape during open processing. If you do not specify a position parameter, the default is 1. For example:

```
filedef fileabc tape1 blp 4
```

positions the tape at the fourth file when it opens fileabc. Because CMS does not know whether files on the tape are label files or data files, the tape is positioned at what is physically the fourth file, regardless of file content. Any label files on the tape are included in counting files.

Label Off (LABOFF) Processing

You should specify LABOFF in the FILEDEF command if you want no positioning or label processing to occur during open processing. The position parameter is not valid for LABOFF. If you specify LABOFF, and your tape is positioned at record 6 in the third file before you issue an OPEN macro, the tape is positioned at exactly the same record after open processing (record 6 in the third file). The following FILEDEF command does not move tape2 (182) before processing the data in fileb:

```
filedef fileb tap2 laboff
```

Nonstandard Label (NSL) Processing

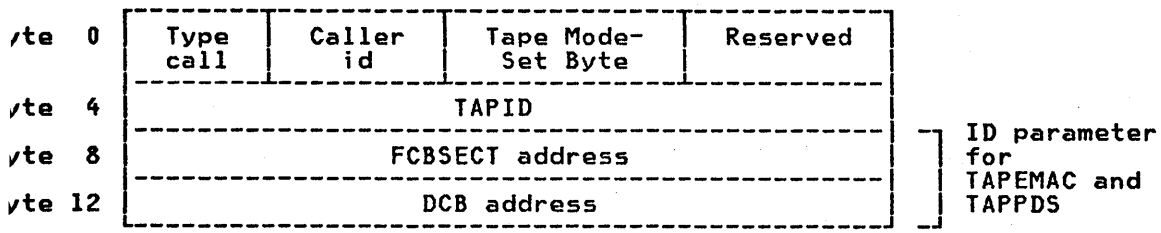
In order to process nonstandard labels, you must write your own routine to read, write, and check the labels. If you have such a routine as a CMS TEXT or MODULE file, you put the filename of the routine after the NSL keyword parameter in the FILEDEF command for the file. The filename must be the name of the first CSECT in the program. It is to this point that control is transferred when the NSL routine gets control. If you do not have a TEXT or MODULE file with the NSL filename you specify, you get an error message. The OPEN and CLOSE routines will load your module if it is not already in storage and will pass control to it at the time they are opening or closing the file. Your routines will then be responsible for processing the tape labels. Nonstandard label routines must do the actual reading and writing of tape labels as well as checking and setting up the label. This is one of several ways nonstandard label processing is different from standard user label processing. Because the CMS label processing routines do not know the size or format of your nonstandard labels, they cannot read or write the labels.

If you use a MODULE file for an NSL routine, it is important that you create the MODULE file so that it starts at an address that will not allow it to overlay the program or command you are executing at the time the NSL routine is invoked.

The reason for this restriction is that the NSL routine is dynamically loaded while your program is executing. For the TAPEMAC and TAPPDS commands, starting the NSL routine at an address above X'21000' prevents such an overlay. If the NSL routine is invoked from your own program which is running in the user area, you must determine how big your program is and where the NSL MODULE file should be located to prevent overlay. Note that you do not have to specify a starting address for NSL routines that are TEXT files. The CMS loader loads such files for you at an address that does not cause an overlay.

Although any user may write his own NSL routine, it is expected that a system programmer will usually write such routines and then other programmers in the installation will use them. Before writing an NSL routine, read the Introduction to CMS, Interrupt Handling, and CMS Functional Information sections in Part 3 of the *VM/SP System Programmers Guide*. In order to ensure proper communication with the CMS system routines, you must use the linkage described below when you write nonstandard label routines.

When an NSL tape label processing routine gets control, register 1 points to a 16-byte parameter list with the following format:



The Type call field is a code telling the type of label processing being done:

```
x'00'     is OPEN input
x'04'     is OPEN output
x'08'     is CLOSE input
x'0C'     is CLOSE output
x'10'     is End Of Tape output
```

The Caller id is a one-byte code which is one of the following:

```
x'80'      Call by OS simulation
x'20'      Call by CMS TAPEMAC or TAPPDS commands
```

Tape modeset byte is used to communicate with the CMS tape I/O routines. It is a one byte hexadecimal code that depends on the type of tape (7 or 9 track), tape density, etc. For further information on the Mode Set, see the TAPE command description in the *VM/SP CMS Command and Macro Reference*. (You probably will pass this byte to the CMS tape controlling module to read and write your tape labels and will never need to know what its codes mean.)

FCBSECT address is the address of the CMSCB (FCBSECT) for the tape file you are processing.

DCB address is the address of the DCB for the tape file you are processing.

> **Note:** For the TAPEMAC and TAPPDS commands, the same interface is used, except that instead of the FCBSECT and DCB address fields, the eight character identifier specified in the ID=identifier field in the

command is passed. This identifier enables you to identify which file you are processing since the TAPEMAC and TAPPDS commands do not work with CMSCBs or DCBs.

Control is passed to your NSL routine by a BALR 14,15 instruction so register 15 contains the address of your routine when you receive control. Register 14 contains the address you should return to when you are finished processing the nonstandard labels. You can return with a BR 14 instruction. When you receive control, register 13 points to a save area in which to store the callers register. The save area linkage is standard OS/VS linkage. You receive control with a PSW key of X'E' which allows you to modify only user storage. When you are finished processing, place a code in register 15 to the CMS label processing routine that called your routine. Place the value 0 (zero) in register 15 if there have been no errors and you want processing to continue normally and the data set to be opened. If you return a nonzero value in register 15, a message is issued to your terminal and the data set is not opened.

If you write the following FILEDEF statement:

```
filedef tapf1 tap1 nsl readlab
```

and have a program called READLAB as a MODULE or TEXT file, your program will receive control when the data set called tapf1 is opened. When your program gets control, register 1 contains the address of the parameter list described above. Using the data in this parameter list, you are able to read or write your own tape header labels. When the same data set is closed, your program again receives control and you can read or write your own trailer labels. Your program can test whether it is getting control for OPEN or CLOSE by examining the type call byte in the parameter list passed to you. If the type call byte is x'10', your NSL routine is being invoked while you are writing an output data set and you have reached the reflective mark that indicates end of tape. You may wish to do special processing in this case. See the "End of Tape" and "End of Volume" section in this publication for further information on end of tape processing.

Differences Between Tape Label Processing Under OS/VS and OS Simulation in CMS

There are a few minor differences in the way CMS OS simulation processes tapes and the way OS/VS processes them. These differences are listed below.

- If you are using OS/VS and you do not specify any label parameter on your JCL statement, the default is SL or standard labels. When you use OS simulation under CMS and do not specify any label information on a FILEDEF statement, the default is LABOFF. LABOFF turns off label processing and nothing is done to position the tape or process labels. Thus, if you specify no label information on FILEDEF, the system will process your tape files exactly the same way they are processed on a CMS system that has no tape label processing facilities.

- You must specify CLOSE to process all trailer labels. No automatic CLOSE occurs at end of data or after reading a tape mark. There is no EOV monitor to process labels before a data set is closed. If an input tape is positioned at an EOF1 or EOV1 record when CLOSE is issued, the label is processed. If a tape file is closed before all data records are read, the trailer label is not processed. Output tapes have EOF records written only at CLOSE time.

- There is no deferred label processing under OS simulation in CMS.

- When the user has not specified a block count routine in his DCB EXIT list under OS/VS, the program abends when a block count error occurs. Under CMS, this condition produces a message that asks whether or not to abend the operation.

- Certain fields in HDR1 and EOF1 labels default to values different from those under OS/VS. These values can always be specified in a LABELDEF command if the user does not like the default values. For example, the default for data set name in an output label under OS simulation is DDNAME and not DSNAME. The default data set sequence number is always one even when the data set is not the first data set on the tape. The default volume sequence number is always one. Read the section on the LABELDEF command in this manual to learn what the default values are under CMS. You can find what default values are in OS/VS by reading the IBM publication *OS/VS Tape Labels*.

  Note: You can always get exactly what you want written on a tape label by explicitly specifying the field on a LABELDEF command. For example, you can specify DSNAME as FID on such a command and have it written in the label instead of DDNAME.

- Default volids (when you do not specify a volid in a LABELDEF or FILEDEF statement) in output HDR1 and EOF1 records under CMS will be CMS001 and will not be the actual volume serial in the VOL1 record already on the tape. It is recommended that you always specify the volid in FILEDEF or LABELDEF to be sure the information written is correct.

- Expiration date specification is always done in absolute form rather than by retention period. You must always use the form yyddd where yy is the year (0-99) and ddd the day (0-366). CMS does not handle expiration dates specified by retention periods.

- When CMS reads a HDR1 label and finds an unexpired file, it always issues a message allowing you to enter "ERROR" or "IGNORE." "ERROR" prevents opening the file in OS simulation. When the DISP MOD option of the FILEDEF command is specified for SL tapes, "IGNORE" allows you to have the tape positioned at the end of the file, ready to add new records. Otherwise, "IGNORE" causes the existing record to be overwritten.

- The NSL routine linkage is quite different under CMS than in OS/VS. (See the section "NSL Processing" for details.)

- Volume serial number verification occurs every time a file on a tape is opened under OS simulation unless the FILEDEF LEAVE option is used for multifile tapes.

- Existing VOL1 labels are not automatically rewritten for density incompatibility in CMS as they are in OS/VS.

- HDR2 records are skipped for input under CMS for OS simulation. They are not checked and information in them is not merged with DCB information. HDR2 records are written (with information obtained from the DCB) on output.

- Blank tapes used for output in CMS cause the tape to run off the reel if you define the tape file as SL or NL. The tape label processing routines try to read

an existing VOL1 or HDR1 label before writing on the tape. Therefore, you should always use the CMS TAPE command to write at least one tape mark (for NL tapes) or a VOL1 label (for SL or SUL tapes) before using the tape to write an output data set.

- If you specify a position parameter that is too big (that is, there are not that many files on the tape), the tape will run off the reel in CMS.
- There are no user exits for user standard labels for EOV label processing in CMS.
- CMS does not support user return codes of 8 and 12 for input standard user labels. If the return code from a user routine is not zero after input label processing, CMS treats it as if the return code was 4. (See the IBM publication *OS/VS1 Data Management Services Guide* or *OS/VS2 MVS Data Management Services Guide* for details).
- No count is kept of user standard labels read or bypassed in CMS. If more than eight such labels exist, the fact is not detected.
- User label processing routines do not receive control under CMS when an abend or a permanent I/O error occurs.
- If a CMS output tape is not positioned at a HDR1 label or a tape mark when label processing begins, error message 422 is issued. Under OS/VS such conditions cause an abend.
- TCLOSE with the REREAD option causes a tape to be rewound under CMS and then forward spaced one file if the tape has standard labels. Under OS/VS, the tape is backspaced four files and forward spaced one file. REREAD for unlabelled tapes in CMS always causes a rewind.

For further information on OS/VS tape label processing, refer to the following IBM publications: *OS/VS1 Data Management Services Guide*, *OS/VS2 MVS Data Management Services Guide*, and *OS/VS Tape Labels*.

For details on end-of-tape/end-of-volume processing under CMS, see the "End-of-Volume" and "End-of-Tape Processing" section later in this publication.

Label Processing in CMS/DOS

You specify the type of label processing you want in CMS/DOS on a DTFMT macro in exactly the same way you specify it when you want to run your program under VSE. See the *VM/SP System Programmer's Guide* for details on CMS support for the DTFMT macro.

Labelled tapes are only supported if you use the DTFMT macro. There is no support for labelled tapes in CMS/DOS for any other type. If you try to read labelled tapes with a DTFCP or DTFDI macro, input standard IBM header labels are skipped, but no other input labels are processed. Output tapes with standard labels have these labels overwritten with a tape mark. All tape work files are treated as output unlabelled files in CMS/DOS although they are defined by a DTFMT. Tapes used for such files have a tape mark written as the first record when the file is opened.

Unlabelled and Nonstandard Labelled Tapes

You define an unlabelled tape with the DTFMT parameter FILABL=NO. The tape file is processed as having no labels.

You define a nonstandard labelled tape with the DTFMT parameter FILABL=NSTD. You also must provide a routine to process your nonstandard labels in the LABADDR=parameter of the DTFMT. Tape processing in CMS for these files is the same as it is under VSE.

Standard Labelled Tapes

You define a standard label tape with the DTFMT parameter FILABL=STD. You also must supply a LABELDEF command to specify label description information. This command replaces the VSE TLBL card and is required for standard label processing under CMS/DOS. The LABELDEF command is discussed in detail in the "LABELDEF Command" section later in this publication.

In order to connect the LABELDEF command for a file with the DTFMT for the same file, you must use the same name to label your DTFMT as you use for a filename in your LABELDEF command. If you code a DTFMT macro in your program as:

```
MT1 DTFMT      ...FILABL=STD
```

you must then supply the following type of LABELDEF command:

```
labeldef mt1 fid yourfile fseq...
```

You can put any description parameters you want on your LABELDEF command but the filename for it must be mt1 if you coded MT1 as the label on the DTFMT.

After you have set up your DTFMT and LABELDEF, you execute your CMS/DOS program. HDR1 labels are checked or written when an OPEN macro is issued. EOF1 labels are checked or written when a CLOSE macro is issued. A VOL1 label volume serial number is checked only if the tape is positioned at load point when the label processing begins and if you have specified a VOLID parameter on a LABELDEF statement for the file. Note, if NOREWIND is not specified in the DTFMT macro for the file, the tape is rewound so it is positioned at load point for label processing.

If you want to process user standard labels as well as standard labels in CMS/DOS, you specify FILABL=STD and also supply a LABADDR parameter in the DTFMT for the file. Control is then transferred to your label processing routines after standard labels are processed. The linkage to user standard label routines is exactly the same as in VSE.

Differences Between Tape Label Processing Under VSE and CMS/DOS

There are minor differences in the way tapes are processed by CMS/DOS and the way they are processed by VSE. These differences are:

- The tape error messages are CMS error messages and not VSE error messages. In some cases VSE allows the system operator to reply NEWTAP to an error message. The system then waits for the operator to mount a new tape and continues processing with this new tape. Such a reply is never possible under

CMS/DOS. In CMS/DOS, you usually can reply IGNORE to ignore a tape label error condition or CANCEL to cancel a job. NEWTAP is never allowed. In a few cases, CMS/DOS allows an IGNORE reply where VSE does not.

- You must specify CLOSE to process all trailer labels. No automatic CLOSE occurs at end of data or after reading a tape mark. If an input tape is positioned at an EOF1 or EOV1 record when CLOSE is issued, the label is processed. If a tape file is closed before all data records are read, the trailer label is not processed. Output tapes have EOF records written only at CLOSE time. For nonstandard labelled tapes, your own routines do not receive control on input when a tape mark is read. You must issue a CLOSE macro in your EOFADDR routine in order to have the trailer labels processed.

- Certain fields in HDR1 and EOF1 labels default to values different from those in VSE. For example, the default volume serial number written in a HDR1 label is CMS001 and not the actual volume serial number (volid) in the VOL1 label already on the tape. The default file sequence and volume sequence numbers are always one even when the file is not the first file on the tape. You should read the section on the LABELDEF command in this publication to learn what the default values are in CMS/DOS. You also can read the IBM publication *VSE/AF Tape Labels* to find what the default values are for VSE. If you do not like the default values, you can always specify the exact values you want in label fields in a LABELDEF command.

- Expiration date specification is always done in absolute form rather than by retention period. You must always use the form yyddd where yy is the year (0-99) and the ddd the day (0-366). CMS does not handle expiration dates specified by retention periods.

- VOL1 labels written in the wrong density are not rewritten automatically by CMS/DOS as they are by VSE.

- Blank tapes should not be used for tape files specified as FILABL=STD in CMS/DOS; they will run off the reel. Use the CMS TAPE command to write a VOL1 label or a tape mark on a blank tape before using it for a STD file.

- Not all tape movement and label checking that occurs in VSE occurs under CMS. For example, when opening an output file, a VSE system expects the tape to be positioned at a HDR1 label or a tape mark. It then backspaces the tape to read the last EOF1 label on the tape. If it does not find the label it expects, it issues an error message. This check is not performed by CMS/DOS. If the tape is not positioned at a HDR1 label or a tape mark when output open processing begins, error message 422 is issued.

- After an EOV1 label is written (see "End-of-Tape/End-of-Volume Processing" later in this publication), the tape is always rewound and unloaded under CMS/DOS. VSE lets a DTFMT parameter control whether or not the tape is rewound.

- User label processing routines do not receive control when an I/O error occurs under CMS/DOS.

- Control is not passed to user standard label routines in CMS/DOS when EOT has been sensed on output and an EOV1 label has been written by the system routines.

- Work tapes are not checked for an expiration date when they contain standard labels under CMS/DOS. If a tape is to be opened as a work tape, CMS/DOS tests to see if it contains a VOL1 label. If it does, a dummy HDR1 label and a tape mark are immediately written on the tape after the VOL1 label. If the tape does not contain a VOL1 label, a tape mark is written at the beginning of the tape. VSE checks expiration dates on previously labelled tapes used as work tapes and gives the operator a chance to reject the tapes if the expiration date has not expired.

For further information on VSE and CMS/DOS tape label processing, refer to the IBM publications, *VSE/AF Tape Labels* and *VSE/AF Macro User's Guide.*

CMS TAPESL Macro

The TAPESL macro is provided for use in CMS programs that do not use OS and DOS simulation features. You can use the CMS TAPESL macro to process IBM standard HDR1 and EOF1 labels without using DOS or OS OPEN and CLOSE macros. You will probably use TAPESL with the RDTAPE, WRTAPE, and TAPECTL macros.

TAPESL processes only HDR1 and EOF1 labels. It does not perform any functions of opening a tape file other than label checking or writing. The TAPESL macro generates linkage to the CMS tape label processing routine that actually processes the label. The macro generates a block of data (32 bytes long) in order to communicate with the tape label processing routines. TAPESL is used both to check and to write tape labels. A LABELDEF command must be issued prior to running the program that contains this macro. The LABID parameter of the TAPESL macro is used to specify the name of the LABELDEF to be used. For example, if you use the macro:

```
TAPESL HOUT,181,LABID=GOODLAB
```

in your assembly language program, you must supply a LABELDEF command for GOODLAB:

```
labedef goodlab fid file10 fseq 4 exdte 78235
```

The tape must be positioned correctly (at the label to be checked or at the place where the label is to be written), before you issue the macro. TAPECTL may be used to position the tape. TAPESL reads or writes only one tape record unless you specify SPACE=YES for input. Then it spaces the tape to beyond the tape mark that ends the label file. TAPESL reads and checks a tape VOL1 label provided the tape is positioned at load point and the user has specified a volid in his LABELDEF command.

Tape Label Processing by CMS Commands

There are three types of CMS commands that do some type of tape label processing. They are:

- TAPEMAC and TAPPDS commands
- TAPE command
- MOVEFILE command

TAPEMAC and TAPPDS Commands

TAPEMAC and TAPPDS have operands where you can indicate the type of label processing you want. The tape must be positioned properly (at the data file or label file you want) before you issue the command. The TAPE command may be used for positioning. A separate LABELDEF command is required for these commands if IBM standard label checking is desired. If SL label type is specified without a labdefid, standard header labels are displayed on the terminal but not checked by the CMS label processing routines. The command:

```
tapemac macfile SL (tap2
```

displays any standard labels that exist on your terminal while the series of commands:

```
labeldef maclab fid macro volseq 2 crdte 77102
tapemac macfile sl maclab (tap2
```

invokes the CMS tape label processing routines. These routines check to see that your tape has a HDR1 label that has a file identifier of macro, a volume sequence number 2, and a creation date of 77102. VOL1 labels are not checked during label processing by TAPEMAC and TAPPDS unless the tape is positioned at load point and you have specified a volid on your LABELDEF command. The DVOL1 function of the TAPE command can be used for volume verification before positioning the tape if the user does not want to start at the first file. These commands process only HDR1 labels; they skip HDR2, UHL, and all trailer labels without processing them.

To process nonstandard tape labels with TAPEMAC and TAPPDS, you use the same interface described in the section "NSL Processing under OS Simulation." The only difference is that instead of putting the CMSCB and DCB addresses in the parameter list, the ID parameter you placed in the command line is passed to your NSL routine.

```
tappds pdsfile cmsut1 * nsl superck id XYZ12345
```

passes the EBCDIC identifier XYZ12345 to your nonstandard label checking routine called SUPERCK. This identifier may be up to eight characters long and is left justified in bytes 8-15 of the parameter list. You can use the identifier to inform your NSL routine of what file you are processing.

Tape Command DVOL1 and WVOL1 Functions

Use the DVOL1 function of the CMSTAPE command to display the VOL1 label of a tape on your terminal. You may use this command to ensure the system operator has mounted the correct tape before you begin processing the tape. If the tape does not have a VOL1 label and you issue the CMSTAPE command, you are informed that the VOL1 label is missing. Do not use TAPE DVOL1 if you have a blank tape. If TAPE DVOL1 is issued and a blank tape is used, CMS will search the entire tape to find the label record; since the tape is void of any records, the tape will run off the end of the reel.

Use the WVOL1 function on the TAPE command to write a VOL1 label on a tape. You can specify a one- to six-character volume serial number (volid) through this command and also a one- to eight-character owner field.

MOVEFILE Command

You can use the MOVEFILE command to move labelled tape files if these files are defined as labelled by the FILEDEF command. The MOVEFILE command supports only SL, NSL, BLP, NL, and LABOFF processing. SUL files are processed as SL files and no user exits are taken.

You can also use the MOVEFILE command to display tape labels on your terminal if you want to see what these labels look like. The following sequence displays the VOL1 and first HDR1 labels on tap4 if the tape has standard labels:

```
filedef in tap4
filedef out term
tape rew (tap4
move in out
```

LABELDEF Command

The LABELDEF command is used to specify the exact data you want written in certain fields of a HDR1 or EOF1 tape label for output. It can also be used to specify fields in the same labels that you want checked on input. If you do not explicitly specify a field for output, a default value is used. If you do not explicitly specify a field for input, the field is not checked. For example:

```
labeldef abc fid master volseq 1 exdte 77364
```

used for input tells CMS to check the file identifier volume sequence number and expiration date in an input HDR1 label. No other fields in the label are checked. The same specification used for output causes the HDR1 label to have MASTER written in the file identifier field, 1 written in the volume sequence number field and 77364 written in the expiration date field. Default values are written in the HDR1 fields that are not specified.

Default values for HDR1 labels are as follows:

FID	for OS simulation, the DDNAME (Specified on FILEDEF) for CMS/DOS, the DTFMT symbolic name for CMS TAPESL macro, the LABELDEF id (LABID=labeldefid) parameter
VOLID	CMS001
VOLSEQ	0001
FSEQ	0001
GENN	blanks
GENV	blanks
CRDTE	current date that label is written
EXDTE	current date that label is written
SEC	0

The filename on the LABELDEF command is used to connect your label definition to a file defined elsewhere. This is why you specify different data for file name

depending on the type of tape label processing you are doing. Filename is DDNAME for OS simulation, DTFMT symbolic name for CMS/DOS and labeldefid for TAPESL.

The LABELDEF command takes the place of the VSE TLBL statement for CMS/DOS.

End-of-Volume and End-of-Tape Processing

There is no true end-of-volume support available with CMS tape label processing. FEOV instructions are not supported under OS simulation and there is no automatic volume switching. Multivolume files are not supported. The following features exist to aid the IBM standard label tape user when he reaches end-of-tape on output or an EOV label in input. These are the only ways in which CMS supports EOV processing.

- Input - When a CLOSE macro is issued or when a TAPESL macro processes an input trailer label, a message is issued if the label read is an EOV1 label instead of an EOF1 label. The EOV1 label is then processed exactly as if it were an EOF1 label. You must request that the operator mount a new tape and reopen a file if you want to continue processing the data.

- Output - Under CMS/DOS and OS simulation processing only (that is, the processing does not occur for TAPESL or CMS commands), the following limited EOV processing occurs:

  1. If you specify that you have an IBM standard label tape file, a single tape mark is written to end your data. This occurs when end-of-tape is sensed on output while you are using regular access method macros to write the file. The tape mark is written immediately after the record that caused the EOT to be sensed. Following this tape mark, CMS writes an EOV1 label and a single tape mark. It then rewinds and unloads your tape. A message is issued telling you that an EOV1 label was written. If you specified nonstandard labels instead of writing the EOV1 label, an exit to the nonstandard label routine you specified for the file is taken after the end-of-data tape mark is written. For BLP or NL files, only the ending tape mark is written.

  2. CMS/DOS jobs are always canceled after an EOT condition is detected on output. In order to continue processing the tape, you must have a new tape mounted, run the same job over again or run a new job and reopen the file.

  3. OS simulation programs that use QSAM or contain a BSAM CHECK macro cause an abend when EOT is detected, with code 001 after an error message. A BSAM program that does not use a CHECK macro has no way of detecting the EOT condition. Such a program continues to try to write on the tape after it is rewound and unloaded. The program enters a wait state rather than continue running to a normal or abnormal completion. Therefore, you should always include a BSAM CHECK macro after the WRITE if you expect your program to reach end-of-tape. OS simulation users are also responsible for completing processing on a new tape with the same or a new job after an EOT is detected.

  4. If you are a CMS/DOS user you always get the automatic output end-of-tape processing described above. However, if you are an OS

simulation user and do not want CMS to do any special end-of-tape processing, you can suppress it by using the NOEOV option on your FILEDEF command for the file. If you enter:

```
filedef dd1 tap3 sl (noeov
```

no tape marks or EOV1 labels are written when EOT is sensed on output. Your tape is not rewound and unloaded. However, the program causes an abend if you use QSAM or include a BSAM CHECK macro after your WRITE macro. Without a CHECK macro, a BSAM program runs the tape off the reel when EOT is sensed and NOEOV is specified.

Error Processing

When the standard label processing routines find errors or discrepancies on tape labels, they send a message to the CMS terminal user who is processing the tape. After an error message is issued, the user can ask the system operator to mount a new tape, use the CMS TAPE command to position the tape at a different file, or respecify his label description information. If you are a terminal user and want another tape mounted, you send the system operator a message telling him what tape to mount.

Some errors cause program termination and others do not. The effect of tape label processing errors depends on both the type of error and the type of program (that is, CMS/DOS, OS simulation, CMS command, etc.) that invokes the label processing. The following are general guidelines on error handling:

- Messages identifying the error are always issued.
- Under OS simulation, tape label errors result in open errors. These errors prevent a tape file from being opened. They do not necessarily end a job. Errors in trailer labels (except block count errors) have no effect on processing.
- In CMS/DOS, the terminal user is generally given two choices: ignore the error or cancel the job. The new-tape option is not allowed.
- The CMS commands TAPEMAC and TAPPDS terminates with a non-zero return code after a tape label error.
- Certain error situations such as unexpired files and block count errors for OS simulation allow the user to ignore the error and do not cause open errors. In these cases, the user enters his decision at the terminal after he is notified of the error.
- Errors that occur during the loading of an NSL routine cause an abend (code 155 or 15A). A block count abend gives an error code of 500.

In all cases, after an error has been detected and diagnosed, you must decide what to do. You may wish to have a new tape mounted and then re-execute the command or you may want to respecify your LABELDEF description if it was incorrect. You can also use the TAPE command to space the tape to a new file if it was positioned incorrectly.

The MOVEFILE Command

The MOVEFILE command can copy sequential tape files into disk files, or sequential disk files onto tape. It can be particularly useful when you need to copy a file from a tape and you do not know the format of the tape.

To use the MOVEFILE command, you must first define the input and output files using the FILEDEF command. For example, to copy a file from a tape attached to your virtual machine at virtual address 181 to a CMS disk, you would enter:

```
filedef input tap1
filedef output disk tape file a
movefile input output
```

This sequence of commands creates a file named TAPE FILE A1. Then use CMS commands to manipulate and examine the contents of the file.

MOVEFILE can also be used to display tape labels and/or move labelled tape files. See "Tape Labels in CMS" for details.

Tapes Created by OS Utility Programs

The CMS command TAPPDS can read OS partitioned and sequential data sets from tapes created by the IEBPTPCH, IEBUPDTE, and IEHMOVE utility programs. When you use the TAPPDS command, the OS data set is copied into a CMS disk file, or in the case of partitioned data sets, into multiple disk files.

IEBPTPCH:

Sequential or partitioned data sets created by IEBPTPCH must be unblocked for CMS to read them. If you have a tape created by this utility, each member (if the data set is partitioned) is preceded with a card that contains "MEMBER=membername." If you read this tape with the command:

```
tappds *
```

then, CMS creates a disk file from each member, using the membername for the filename and assigning a filetype of CMSUT1. If you want to assign a particular filetype, for example TEST, you could enter the command as follows:

```
tappds * test
```

If the file you are reading is a sequential data set, you should use the NOPDS option of the TAPPDS command:

```
tappds test file (nopds
```

The above command reads a sequential data set and assigns it a file identifier of TEST FILE. If you do not specify a filename or filetype, the default file identifier is TAPPDS CMSUT1.

IEBUPDTE:

Tapes in control file format created by the IEBUPDTE utility program can be read by CMS. Data sets may be blocked or unblocked, and may be either sequential or partitioned. Since files created by IEBUPDTE contain ./ADD control cards to signal the addition of members to partitioned data sets, you must use the COL1

option of the TAPPDS command. Also, you must indicate to CMS that the tape was created by IEBUPDTE. For example, to read a partitioned data set, you would enter the command:

```
tappds * test (update col1
```

The CMS disk files created are always in unblocked, 80-character format.

IEHMOVE:

OS unloaded partitioned data sets on tapes created by the IEHMOVE utility program can be read either by the TAPPDS command or by the TAPEMAC command. The TAPPDS command creates an individual CMS file from each member of the PDS.

If the PDS is a macro library, you can use the TAPEMAC command to copy it into a CMS MACLIB. A MACLIB, a CMS macro library, has a special format and can usually be created only by using the CMS MACLIB command. If you use the TAPPDS command, you have to use the MACLIB command to create the macro library from individual files containing macro definitions.

Specifying Special Tape Handling Options

For most of the tape handling that you do in CMS, you do not have to be concerned with the density or recording format of the magnetic tapes that you use. There are, however, some instances when it may be important and there are command options that you can use with the TAPE command MODESET operand and with ASSGN and FILEDEF command options.

The specific situations and the command options you should use are listed below.

- If you are reading or writing a 7-track tape and the density of the tape is either 200 or 556 bpi, you must specify DEN 200 or DEN 556.
- If you are reading or writing a 7-track tape with a density of 800 bpi, you must specify 7TRACK.
- If you are reading or writing a 7-track tape without using the data convert feature, you must use the TRTCH option.
- If you are writing a tape using a 9-track dual density tape drive with the 9TRACK option specified, and you want the density to be 800 (on an 800/1600 drive) or 6250 (on a 1600/6250 drive), then you must specify DEN 800 or DEN 6250.
- If you are writing a tape, the default tape block size is 4096 bytes plus a 5-byte header. This format is not compatible with previous VM/370 systems. Therefore, if you want to write a tape compatible with previous VM/370 systems, you must use the "BLKSIZE 800" option of the TAPE command. The TAPE command is described in detail in *VM/SP CMS Command and Macro Reference*.

Chapter 7. Communicating with Other Computer Users

Using CMS commands, you are able to send information (files, messages, and notes) to other computer users and to receive information from them. You can collect the necessary information about other computer users with whom you communicate to keep in your "userid NAMES" file. The following CMS commands reference the NAMES file created via the CMS NAMES command:

NAMEFIND	Display/Stack information from a NAMES file.
NOTE	Prepare a "note" for one or more computer users, to be sent via the SENDFILE command.
RECEIVE	Read onto disk, a file or note that is in your virtual reader.
SENDFILE	Send files or notes to one or more users on your system or a system that is attached to yours via Remote Spooling Communications Subsystem (RSCS) by issuing the command or by using a menu (display terminal only).
TELL	Send a message to one or more computer users who are logged on to your computer or to one attached to yours via RSCS.

What is a Names File?

A names file is a collection of information about other users with whom you communicate. Having a names file makes it easier for you to communicate with others because you can assign nicknames to them. An "entry" in a names file contains all of the information associated with a a particular nickname that you enter on one menu. You can also create an entry for a list of names, where the nickname would refer to the whole list.

Creating a Names File

The first entry in your names file, should be for yourself. The information will be used for note headings, which are discussed later on. To display the names screen, enter:

```
names
```

When you enter NAMES, your userid appears (automatically) in the first line. The following is an entry in the file "ZOOKEEP NAMES."

```
 ====>  ZOOKEEP NAMES    <========>  N A M E S   FILE   EDITING  <====

Fill in the fields and press a PFkey to display and/or change your NAMES file

Nickname:  ZOO      Userid: ZOOKEEP  Node: CITYZOO   Notebook:
                      Name: Zoo Keeper
                     Phone: 123-4567
                   Address: City Zoo
                          :
                          :
                          :
              List of Names:
                          :
                          :
                          :

You can enter optional information below. Describe it by giving it a ''tag.''

Tag:                  Value:
Tag:                  Value:

1= Help     2= Add      3= Quit     4= Clear        5= Find     6= Change
7= Previous 8= Next     9=          10= Delete      11=         12= Cursor

====>
                                                          MACRO-READ 1 FILE
```

Figure 7-1. Sample NAMES Screen

Entering a List of names

The list of names is something like a distribution list. If you send notes, files, or messages to groups of people, you can create an entry in your names file for each group. In this case, the nickname represents the name that you want to call this list. You can specify the names of the people in the list in the following ways:

- as a nickname of an entry in the names file;
- as a userid of a user who shares your computer;
- in the form "userid AT node."

Each time you send a note, a file, or a message to the nickname specified, it will go to everyone on this list. The following menu shows an entry for a list of names. Each name in the list is the nickname of an entry in the names file.

```
 ====>  ZOOKEEP NAMES    <========>  N A M E S   FILE  EDITING  <====
Fill in the fields and press a PFkey to display and/or change your NAMES file

Nickname: ANIMALS  Userid:          Node:          Notebook:
                     Name:
                    Phone:
                  Address:
                          :
                          :
                          :
            List of Names:  BEAR LION MONKEY
                          :
                          :
                          :

You can enter optional information below. Describe it by giving it a ''tag.''

Tag:                Value:
Tag:                Value:

1= Help     2= Add     3= Quit     4= Clear        5= Find     6= Change
7= Previous 8= Next    9=         10= Delete      11=         12= Cursor

====>
                                                          MACRO-READ 1 FILE
```

Figure 7-2. Sample Entry for a List of Names.

Entering Chained Lists of Names

Use chained Lists of Names, to allow many users to be included in the List of Names tag. For this example, there is an entry in the ZOOKEEP NAMES file called BIRDS containing a List of Names as shown in Figure 7-3 .

```
 ====>  ZOOKEEP NAMES    <========>  N A M E S   FILE  EDITING  <====
Fill in the fields and press a PFkey to display and/or change your NAMES file

Nickname: BIRDS    Userid:          Node:          Notebook:
                     Name:
                    Phone:
                  Address:
                          :
                          :
                          :
            List of Names:  OWL SWAN TURKEY
                          :
                          :
                          :

You can enter optional information below. Describe it by giving it a ''tag.''

Tag:                Value:
Tag:                Value:

1= Help     2= Add     3= Quit     4= Clear        5= Find     6= Change
7= Previous 8= Next    9=         10= Delete      11=         12= Cursor

====>
                                                          MACRO-READ 1 FILE
```

Figure 7-3. Another Sample Entry for a List of Names.

Figure 7-4 shows how BIRDS and ANIMALS can be represented by two nicknames. Each name in the list is the nickname of an entry in the names file.

```
 ====>  ZOOKEEP NAMES    <========>  N A M E S   FILE  EDITING  <====
Fill in the fields and press a PFkey to display and/or change your NAMES file

Nickname: BOARDERS Userid:          Node:          Notebook:
                    Name:
                   Phone:
                 Address:
                       :
                       :
                       :
           List of Names:  ANIMALS BIRDS
                       :
                       :
                       :

You can enter optional information below. Describe it by giving it a ''tag.''

Tag:                Value:
Tag:                Value:

1= Help     2= Add     3= Quit     4= Clear        5= Find     6= Change
7= Previous 8= Next    9=         10= Delete      11=         12= Cursor

====>
                                                          MACRO-READ 1 FILE
```

Figure 7-4. Sample Entry for a Chained List of Names.

If a note is sent to BOARDERS, the following receive the note:

TO NICKNAME	CHAINED LIST OF NAMES	ACTUAL RECIPIENTS
BOARDERS	ANIMALS	BEAR
		LION
		MONKEY
	BIRDS	OWL
		SWAN
		TURKEY

The following represents the ZOOKEEP NAMES file:

```
:nick.ZOO       :userid.ZOOKEEP  :node.CITYZOO
                :name.Zoo Keeper                  :phone.123-4567
                :addr.City Zoo
:nick.BEAR      :userid.GRIZZLY  :node.DEN
                :name.I. M. Grizzley              :phone.123-4567
                :addr.Den;City Zoo
:nick.LION      :userid.COWARD   :node.DEN
                :name.I.M.A. COWARD               :phone.123-4567
                :addr.Lion Den;City Zoo
:nick.MONKEY    :userid.MONKEY   :node.TREE
                :name.T.O.P. Banana               :phone.123-4567
                :addr.Banana Tree;City Zoo
:nick.ANIMALS
                :list.BEAR LION MONKEY
:nick.OWL       :userid.OWL      :node.TREE
                :name.I. M. Wise                  :phone.123-4567
                :addr.Big Tree;City Zoo
:nick.SWAN      :userid.SWAN     :node.SWANLAKE
                :name.Grace Full                  :phone.123-4567
                :addr.Swan Lake;City Zoo
:nick.TURKEY    :userid.TURKEY   :node.COTTAGE
                :name.T.TURKEY                    :phone.123-4567
                :addr.Turkey Coop;City Zoo
:nick.BIRDS
                :list.OWL SWAN TURKEY
:nick.BOARDERS
                :list.ANIMALS BIRDS
```

Figure 7-5. Sample 'userid NAMES' File

Sending Messages

You can send messages to one or more users on your computer or on other computers that are connected to yours via the Remote Spooling Communications Subsystem (RSCS) network. The users must be logged on to receive your message. Because the TELL command references your names file, you are able to use nicknames. For example:

```
tell bear There is honey for dessert!
```

If Bear is logged on, he sees the following message on his screen.

```
MSG FROM ZOOKEEP :  There is honey for dessert!
```

You can send a message to a list of people when you have a nickname for the list. For example:

```
tell animals Good Morning!
```

sends the message to the list of names for the nickname ANIMAL (BEAR LION, and MONKEY). They must be logged on to receive your message.

You can also use the CP MESSAGE command to send a message to a user or to the primary system operator. The recipient must be logged on to receive your message. Use the CP QUERY userid command to see if another user is logged on. The CP MESSAGE command does not use your names file. An example of the CP MESSAGE command using the abbreviation (MSG) is:

```
cp msg jonescj we are leaving now.
```

Receiving Messages

During the course of a terminal session, you can receive many kinds of messages from VM/SP, from the system operator, from other users, or from your own programs. You can decide whether or not you want to receive these messages. For example, if you use the command:

```
cp set msg off
```

You will not receive any messages sent by the TELL command or the CP MESSAGE command; if another virtual machine user tries to send you a message, he receives the message:

```
userid NOT RECEIVING, MSG OFF
```

If your virtual machine handles special messages and you do not want to receive special messages at this time, you can issue:

```
cp set smsg off
```

You will not receive any special messages sent by the CP SMSG command; if another virtual machine user attempts to do so, he receives a message:

```
userid NOT RECEIVING, SMSG OFF
```

Similarly, to prevent warning messages (which usually come from the system operator) from coming to you, you can use:

```
cp set wng off
```

However, you would only do this in cases where you were typing some output at your terminal and did not want the copy ruined.

VM/SP issues error messages whenever you issue a command incorrectly or if a command or program fails. These messages have a long form, consisting of the error message code and number, followed by text describing the error. If you wish to receive only the text portion of messages with severity codes I, E, and W (for informational, error, and warning, respectively), you can issue the command:

```
cp set emsg text
```

If you want to receive only the message code and number (from which you can locate an explanation of the error in *VM/SP System Messages and Codes*), you specify:

```
cp set emsg code
```

You can also cancel error messages completely:

```
cp set emsg off
```

To restore the EMSG setting to its default, which is the message text, enter:

```
cp set emsg text
```

Some CP commands issue informational messages telling you that CP has performed a particular function. You can prevent the reception of these messages with the command:

```
cp set imsg off
```

or restore the default by issuing:

```
cp set imsg on
```

The setting of EMSG applies to CMS commands as well as to CP commands.

You can also control the format of the CMS ready message. If you enter:

```
set rdymsg smsg
```

you receive only the "R;" or shortened form of the ready message after the completion of CMS commands. If you are not receiving error messages (as described above) and an error occurs, the return code from the command still appears in parentheses following the "R."

Sending Notes and Files

When you have short communication, like a letter, use the NOTE command to prepare a "note" to send to one or more users on your computer or on other computers that are connected to yours via the Remote Spooling Communications Subsystem (RSCS) network. The person to whom you are sending the note need not be logged on. The NOTE command references your "userid NAMES" file, so you can use nicknames when you create your notes.

Composing Notes

Entering NOTE name puts you in XEDIT mode. Enter the INPUT subcommand on the command line and type the body of the note in the space provided. Press the PF2 key to add lines if you need more space. For example:

```
note bear
```

results in a screen as in Figure 7-6 where the body of the note was entered in the space provided.

```
  ZOOKEEP NOTE     A0  F 80  TRUNC=80 SIZE=24 LINE=12 COL=1 ALT=0
OPTIONS: NOACK  LOG  SHORT  NOTEBOOK ALL

Date: 11 February 1981, 11:04:52 EDT
From: Zoo Keeper                123-4567            ZOOKEEP  at CITYZOO
To:   BEAR at DEN



Dear Bear,
    I have some good news.  Someone ordered 500 jars of honey.
You can have honey for dessert all month if you like.



                                        Sincerely,
                                        Zoo Keeper

1= Help     2= Add line 3= Quit    4= Tab          5= Send      6= ?
7= Backward 8= Forward  9= =      10= Rgtleft     11= Spltjoin 12= Power input

====>
                                                          X E D I T  1 FILE
```

Figure 7-6. Sample Note with Short Headings

Sending a Note

To send the note, you can do *one* of the following:

- Press the PF5 key.
- Enter on the command line, SENDFILE (NOTE or SENDFILE (NOTE OLD

  Note: Use the OLD option when recipients do not have the RECEIVE command available to them so that the file can be DISK LOADed.

- Enter SEND (a synonym for "SENDFILE (NOTE").

Continuing a Note

If you want to save a note and finish it later, enter FILE on the command line. The note will *not* be sent. It is saved on your disk as "userid NOTE A0." To continue the note later on, issue the NOTE command *with no parameters.*

Keeping a Copies of Notes

Each time that you send a note, a copy is kept in a file called ALL NOTEBOOK. A note is saved by appending it to the NOTEBOOK file. Notes are separated by a line of 73 equal signs (=). If you want to keep separate notebooks for certain correspondence, you can:

- Specify a notebook filename with the NOTE command. For example, to save a note in ANIMALS NOTEBOOK, enter:

  ```
  note bear (notebook animals
  ```

- Specify a notebook filename on an entry in your "userid NAMES" file.
- Set up a default notebook filename with the DEFAULTS command. Notes will go into this notebook unless you have specified a notebook filename with the NOTE command or if you have specified a notebook filename on the recipient's entry in your names file. For example to set up the default to save notes in ANIMALS NOTEBOOK, enter:

  ```
  defaults note notebook animals
  ```

See the *VM/SP CMS Command and Macro Reference* for information about the DEFAULTS command.

Sending Files

Use the SENDFILE command to send files and notes to one or more computer users on your computer or on other computers that are connected to yours via the Remote Spooling Communications Subsystem (RSCS) network. Since SENDFILE is one of the commands that references your names file, you can use nicknames to identify the recipients. The nickname is automatically converted into node and userid.

If you know the name of the file that you want to send, you can just enter the file identification and nickname following the SENDFILE command. For example:

```
sendfile banana split a to monkey
```

Otherwise, if you cannot remember the name of the file or if you have many files to send, enter SENDFILE without operands. When you enter the SENDFILE command (or the abbreviation SF), a special screen is displayed.

The following is a sample SENDFILE menu:

```
                  ---------------  SENDFILE  ---------------
File(s) to be sent     (use * for Filename, Filetype and/or Filemode
                            to select from a list of files)
Enter filename : *
      filetype : data
      filemode : a

Send files to  : monkey

Type over YES or NO to change the options:

   NO     Request acknowledgement when the file has been received?

   YES    Make a log entry when the file has been sent?

   YES    Display the file name when the file has been sent?

   NO     This file is actually a list of files to be sent?


  1= Help          3= Quit             5= Send            12= Cursor

====>
                                                      MACRO-READ 1 FILE
```

Figure 7-7. Sample SENDFILE Menu

In Figure 7-7, the sender entered an asterisk for filename, "data" for filetype, and "a" for filemode. The name of the recipient (MONKEY) is also entered on the screen. When PF5 (or the ENTER key) is pressed, a special FILELIST screen is displayed. The files to be sent can be selected from this screen (shown in Figure 7-8).

```
  ZOOKEEP FILELIST    A0  V 108  TRUNC=108 SIZE=418 LINE=1 COL=1 ALT=0

Cmd Filename Filetype Fm Format Lrecl Records Blocks   Date    Time
    ANIMAL   DATA     A1 V         95      34      2 10/04/80 21:12:04
 s  BEAR     DATA     A1 V         95      29      2 10/04/80 20:58:07
    CAMEL    DATA     A1 V        107     281     10 10/04/80 17:59:00
 s  LION     DATA     A1 V         92     101      4 10/02/80 15:33:05
    MYSTERY  DATA     A2 V         75      28      1  9/25/80 12:10:03
 s  MONKEY   DATA     A2 V        120     277     10  9/24/80  9:14:02
    SWAN     DATA     A1 V         26       7      1  9/23/80 16:50:06
    ZOO      DATA     A1 V         80     489     30  8/26/80 16:05:08
1= Help     2= Refresh  3= Quit    4= Sort(type) 5= Sendfile   6= Sort(size)
7= Backward 8= Forward  9= FL /n  10=           11= XEDIT     12= Cursor
 Type 'S' in front of each file to be sent, and press ENTER.
====>
                                                      XEDIT 1 FILE
```

Figure 7-8. Sample FILELIST Screen Invoked from SENDFILE

To send one or more of these files, you can type a letter "s" in front of the filename of each file you want sent (see above) and then press the ENTER key. You can also position the cursor on the line describing the file you want to send, and then press the PF5 key.

Sending One File

To send only one file:

1. Enter SENDFILE (or its abbreviation SF).
2. On the SENDFILE menu, enter the filename, filetype and filemode in the spaces provided. If the filemode is A, you can leave filemode blank.
3. Enter the names of the recipient(s). Remember that you can use nicknames.
4. Select the "YES/NO" options.
5. Press either PF5 or the ENTER key to send the file. Pressing:
   - PF5 sends the file and exits from the menu.
   - the ENTER key sends the file and keeps the menu.

Receiving Notes and Files

After you logon you might see a message notifying you that you have files in your reader. For example:

```
FILES:  004 RDR, NO PRT, NO PUN
```

During your terminal session if you want to find out if you have files in your virtual reader you can enter any of the following commands:

RDRLIST	Displays information about the files in your virtual reader with the ability to issue commands from the list.
RDR	Generates a return code and either displays or stacks a message that identifies the characteristic of the next file in your reader.
CP QUERY RDR ALL	Lists your reader files (if any) and their characteristics.
CP QUERY FILES	Displays the number of spool files in your virtual machine.

For example, when there are no files in your reader:

If you enter:	**The system responds:**
rdrlist	No files in your reader. R(00028);

If you enter:	The system responds:
rdr	READER EMPTY R;
cp q rdr all	NO RDR FILES R;
cp q files	FILES: NO RDR, NO PRT, NO PUN R;

Using the CMS RDRLIST Command

Entering the CMS RDRLIST command give you a display about the files in your reader. For example:

```
 OHARA    RDRLIST     A1  V 108  TRUNC=108 SIZE=17 LINE=1 COL=1 ALT=1
Cmd      Filename Filetype Class User At Node Hold  Records  Date  Time
         PIZZA    TOPPINGS PUN A KEN   NODE04 NONE       10  10/06 10:39:38
         COOKIE   ASSEMBLE PUN A KEN   NODE04 NONE       10  10/06 10:25:11
         $JELLY   NOTE     PRT A KEN   NODE04 NONE        7  10/06 10:15:50
         DIETING  TIPS     PUN A KEN   NODE04 NONE       11  10/06 09:40:28
         KEN      NOTE     PUN A KEN   NODE04 NONE       10  10/06 08:43:07
         SEND     EXEC     PUN A BOB   NODE02 NONE        2  10/06 07:12:35
         GOOD     DAY      PUN A GEOFF NODE02 NONE       29  10/05 11:44:34
         Acknowl  edgment  PUN A BOB   NODE02 NONE        2  10/05 11:42:21

1=Help      2=Refresh   3=Quit      4=Sort(type) 5=Sort(date) 6=Sort(user)
7=Backward  8=Forward   9=Receive  10=          11=Peek      12=Cursor

====>
                                                                    XEDIT
```

Figure 7-9. Sample RDRLIST Screen

Some of the commands that you can issue from the list are:

PEEK	Displays a file in your virtual reader without reading it onto disk.
RECEIVE	Reads onto disk a file or note that is in your virtual reader.
DISCARD	Purges a file displayed in the reader list.

Receiving a File

If you have issued the RDRLIST command and you want to receive a file:

1. Move the cursor to the line describing the file that you want to receive.
2. Press PF9. A notice will appear on that line, telling you that the file has been received.

For example:

```
 OHARA     RDRLIST      A1  V 108  TRUNC=108 SIZE=17 LINE=1 COL=1 ALT=1
Cmd      Filename Filetype Class User At Node Hold  Records  Date  Time
         PIZZA    TOPPINGS PUN A KEN   NODE04 NONE       10  10/06 10:39:38
*        COOKIE   ASSEMBLE recv from Ken at NODE04
         $JELLY   NOTE     PRT A KEN   NODE04 NONE        7  10/06 10:15:50
         DIETING  TIPS     PUN A KEN   NODE04 NONE       11  10/06 09:40:28




1=Help      2=Refresh   3=Quit      4=Sort(type) 5=Sort(date) 6=Sort(user)
7=Backward  8=Forward   9=Receive  10=          11=Peek      12=Cursor

====>
                                                                    XEDIT
```

Figure 7-10. Sample RDRLIST Screen after Receiving a File

If the file in your reader has the same name as a file that is already on your disk, after you press PF9, you will receive the following message on your RDRLIST screen:

```
File 'fn ft fm' already exists.--specify 'REPLACE' option.
```

If you want to replace the file on your disk, then enter the following in the "Cmd" space next to the file that you want to receive:

```
receive /(replace
```

and then press the ENTER key. The file on your disk will be replaced.

If you want to keep the file on your disk, you can either:

- rename the file on your disk (use the CMS RENAME command).

or

- give the file to be read in a new name. For example, enter the following in the "Cmd" space and press the ENTER key:

  ```
  receive / banana split
  ```

Receiving a Note

You can receive notes in the same way that you receive other types of files. A note will have a filetype of NOTE. Just move the cursor to the line describing the note and press PF9.

The note is appended to your ALL NOTEBOOK file, unless you have a notebook specified in your names file entry for that person. For example, after receiving $JELLY NOTE, the sample screen would look like the following:

```
 OHARA    RDRLIST     A1  V 108  TRUNC=108 SIZE=17 LINE=1 COL=1 ALT=1
Cmd     Filename Filetype Class User At Node Hold  Records  Date  Time
        PIZZA    TOPPINGS PUN A KEN   NODE04 NONE       10  10/06 10:39:38
*       COOKIE   ASSEMBLE recv from Ken at NODE04
*       $JELLY   NOTE     added to ALL NOTEBOOK A0
        DIETING  TIPS     PUN A KEN   NODE04 NONE       11  10/06 09:40:28



1=Help      2=Refresh   3=Quit      4=Sort(type) 5=Sort(date) 6=Sort(user)
7=Backward  8=Forward   9=Receive  10=          11=Peek      12=Cursor

====>
                                                                    XEDIT
```

Figure 7-11. Sample RDRLIST Screen after Receiving a Note

Discarding a File

Use the DISCARD command to purge a file displayed in your RDRLIST file. Unlike the CP PURGE command, DISCARD allows an acknowledgment to be sent to the sender (if one was requested). The acknowledgment indicates that the file was discarded. DISCARD also makes an entry in your "userid NETLOG" file, (if the log option was in effect in the RECEIVE command) indicating that the file was discarded. To discard a file, enter DISCARD on the "Cmd" space next to the file that you want to discard and press the ENTER key.

Loading and Purging files

Your response from the RDR command might indicate that the file was DISK DUMPed to you. For example, the response might be:

```
DISK LOAD TEST1 SCRIPT A1
R(00022);
```

You can enter DISK LOAD to read the file onto your disk.

If after doing a QUERY RDR ALL, you know that you want to get rid of files in your reader, then you can use the CP PURGE command. For example, if you want to purge a reader file with a spoolid of 1234, enter:

```
cp purge rdr 1234
```

If you want to purge all reader files in a certain class, for example, all Class A files, then you might enter:

```
cp purge rdr cl a
```

Alternate Method of Sending Files

You can send printer, punch, or reader spool files to other users. To send a spool file, you must know the userid of the virtual machine at your location that is running RSCS and the location identification (locid) of the remote location. If you are sending a spool file to a particular user at the remote location, you should also know that userid of the user.

The CP commands that you can use to transmit files across the network are TAG and SPOOL.

- The TAG command allows you to specify the locid and userid of the virtual machine that is to receive a spool file, or, in the case of tagging a printer or punch, of any spool files produced by that device.
- With the SPOOL command, you spool your virtual device to the RSCS virtual machine.

An example of using the CP commands SPOOL, TAG, and the CMS DISK command to send someone a file is:

```
cp sp pun net4
cp tag dev pun node04 murphybe
disk dump brian file a1
```

where net4 is the rscsid of the virtual machine, node4 is the locid, and murphybe is the userid of the recipient.

After you disk dump the file, You should spool your punch back to yourself.

To spool your punch back to yourself, enter:

```
sp pun *
```

The SENDFILE command makes it easier for you to send files to others since you do not have to use the CP SPOOL and TAG commands each time you send a file. Refer to the documentation about sending and receiving files earlier in this chapter.

You can also use the CP TRANSFER command to transfer files from your own virtual reader.

For information on the CP commands SPOOL, TAG, and TRANSFER, refer to the *VM/SP CP Command Reference for General Users.*

Part 2: Program Development Using CMS

You can use CMS to write, develop, update, and test:

- CMS programs to execute in the CMS environment
- OS programs to execute either in the CMS environment (using OS simulation) or in an OS virtual machine
- VSE programs to execute in either the CMS/DOS environment or in a VSE virtual machine

As you learn to use CMS, you may want to write programs for CMS applications. Chapter 8, "Programming for The CMS Environment" contains information for assembler language programmers: linkage conventions, programming notes, and macro instructions you can use in CMS programs.

The OS and VSE simulation capabilities of CMS allow you to develop OS and VSE programs interactively in a time-sharing environment. When your programs are thoroughly tested, you can execute them in an OS or VSE virtual machine under the control of VM/SP.

Chapter 9, "Developing OS programs under CMS" is for programmers who use OS. It describes procedures and techniques for using CMS commands that simulate OS functions.

Chapter 10, "Developing VSE Programs Under CMS" is for programmers who use VSE. It describes procedures and techniques for using CMS/DOS commands to simulate VSE functions.

If you use VSAM and Access Method Services in either a VSE or an OS environment, Chapter 11, "Using Access Method Services and VSAM Under CMS and CMS/DOS" provides usage information for you. It describes how to use CMS to manipulate VSAM disks and data sets.

The CMS batch facility is a CMS feature that allows you to send jobs to another machine for execution. How to prepare and send job streams to a CMS batch virtual machine is described in Chapter 12, "Using the CMS Batch Facility."

You can use the interactive facilities of CP and CMS to test and debug programs directly at your terminal. Chapter 13, "Debugging Your Program Using VM/SP" shows examples of commands and debugging techniques.

Chapter 8. Programming for The CMS Environment

This section contains information for assembler language programmers who may need to write programs to be used in the CMS environment. The conventions described here apply only to CMS virtual machines; you can not execute these programs under any other operating systems.

Program Linkage

Program linkages, in CMS, are generally made by means of a supervisor call instruction, SVC 202. The SVC handling routine takes care of program linkage for you. The registers used and their contents are discussed in the following paragraphs.

Register	Contents
0	If the command is called from the terminal or from an EXEC 2 EXEC, register 0 points to an extended plist, which contains addresses referring to the extended command as it was initially entered by the user.
1	Points to a parameter list of successive doublewords. The first entry in the list is the name of the called routine or program, and any successive doublewords may contain arguments passed to the program. Parameter lists are discussed under "Parameter Lists."
13	Contains the address of a 24-fullword save area, which you can use to save your caller's registers. This save area is provided to satisfy standard OS and DOS linkage conventions; you do not need to use it in CMS, since the SVC routines save the registers.
14	Contains the return address of the SVC handling routines. You must return control to this address when you exit from your program. The CMS routines that get control by way of register 14 close files, update your disk file directory, and calculate and type the time used in program execution. These values appear in the CMS ready message, which is displayed at your terminal when your program finishes execution: `R;T=n.nn/x.xx hh:mm:ss` *where:* `n.nn` is the CMS CPU time (in seconds) `x.xx` is the combined CP and CMS CPU time. `hh:mm:ss` is the time of day in hours, minutes, and seconds. **Note:** If CMS cannot calculate a valid time, it will display *.** in place of n.nn/x.xx. If the CMS CPU time or the combined CP and CMS CPU time exceeds 35 minutes, the printed time may be incorrect.

12 and 15 Contain your program's entry point address. You can use this address to establish immediate addressability in your program. You should not use Register 15 as a base address, since all CMS SVCs use it for communication with your programs.

Figure 8-1 on page 8-3 shows a sample CMS assembler language program entry and exit.

Return Code Handling

Register 15, in addition to its role in entry linkage, is also used in CMS as a return code register. All of the CMS internal routines pass a completion code by way of register 15, and the SVC routines that receive control when any program completes execution examine register 15.

If register 15 contains a nonzero value, this value is placed in the CMS ready message, following the "R":

```
R(nnnnn);T=n.nn/x.xx hh:mm:ss
```

When you are executing programs in CMS, it is good practice, if your programs do not use register 15 as a return code register, to place a zero in it before transferring control back to CMS. Otherwise, the ready message may display meaningless data.

Parameter Lists

When you enter a command at your terminal, a CMS scan routine sets up two distinct parameter lists based on your command input line.

The first type of parameter list created is known as a tokenized parameter list. It is doubleword aligned, with parameters occupying successive doublewords. The scan routine recognizes blanks and parentheses as argument delimiters; parentheses are placed, in the parameter list, in separate doublewords. If you enter an argument longer than eight characters, the argument is truncated and only the first eight characters of the argument will appear in the parameter list. However, no error condition results. General purpose register 1 (R1) contains the address of this parameter list.

The second type of parameter list that is created is known as an extended (untokenized) parameter list. It consists of four addresses that indicate the extended form of the command as it was entered at the terminal. The first non-blank character, left parenthesis, or right parenthesis following the command is treated as a delimiter to determine where the pointer to the start of the argument is. General purpose register 0 (R0) contains the address of this parameter list.

For example, if you have a CMS MODULE file named TESTPROG, and you call it from the command line as follows:

```
testprog(file2)
```

The scan routine sets up the following tokenized parameter list

```
CMNDLIST DS    0D
         DC    CL8'TESTPROG'
         DC    CL8'('
         DC    CL8'FILE2'
         DC    CL8')'
         DC    8X'FF'
```

```
PROGRAM  CSECT
         USING PROGRAM,12       ESTABLISH ADDRESSABILITY
         ST    14,SAVRET        SAVE RETURN ADDRESS IN R14
          .
          .
          .
         L     14,SAVRET        LOAD RETURN ADDRESS
         LA    15,0             SET RETURN CODE IN R15
         BR    14               GO
SAVRET   DS    F                SAVE AREA
```

Figure 8-1. Sample CMS Assembler Program Entry and Exit Linkage

The last doubleword consists of all X'F's. This acts as delimiter to indicate the end of the parameter list.

The scan routine also sets up the following extended parameter list:

```
EPLIST DC    A(CMDSTART)
       DC    A(ARGSTART)
       DC    A(ARGEND)
       DC    A(0)
```

The extended parameter list refers to the same command line in the following way:

```
CMDSTART DC  C'testprog'
ARGSTART DC  C'(file2)'
ARGEND   EQU *
```

The left parenthesis following 'testprog' is the delimiter to determine ARGSTART. The following is another example of how the extended parameter list is set up. If you called TESTPROG from the command line:

```
testprog  file2
```

The scan routine sets up the tokenized parameter list:

```
CMNDLIST DS    0D
         DC    CL8'TESTPROG'
         DC    CL8'FILE2'
         DC    8X'FF'
```

The extended parameter list is set up the following way:

```
CMDSTART DC  C'testprog
ARGSTART DC  C'file2'
ARGEND   EQU *
```

The first non-blank character following 'testprog' is the delimiter that determines ARGSTART.

If you do not specify any arguments and have called TESTPROG from the command line as follows:

```
testprog
```

The scan routine sets up the tokenized parameter list:

```
CMNDLIST DS     0D
         DC     CL8'TESTPROG'
         DC     8X'FF'
```

The extended parameter list will be:

```
CMDSTART DC  C'testprog'
ARGSTART DC  *
ARGEND   EQU *
```

When there are no arguments, ARGSTART is set equal to ARGEND.

Using Parameter Lists

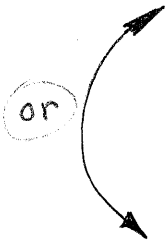
The scan routine that sets up the parameter lists places the address of the lists in R0 and R1, as previously specified, and then calls the SVC handling routine. The SVC routine gives control to the program named in the first doubleword of the tokenized parameter list.

When your program receives control, it can examine the parameter list passed to it by way of either R0 or R1, or both.

You can use this technique, also, to call CMS commands from your programs.

When you use the LOAD and RUN commands to execute a program in CMS, you can pass an argument list to the program on the command line. For example, if you enter:

```
load myprog
start * run1 proga
```



the "*" indicates that the entry point is to be defaulted. The arguments RUN1 and PROGA are placed in a parameter list of doublewords and register 1 contains the address of this list when your program receives control. If you want to use the RUN command to perform the load and start functions, you could enter:

```
run myprog (run1 proga
```

"run" is an EXEC

The parenthesis indicates the beginning of the argument list.

To detect the absence of a parameter list that occurs when the LOAD command START option is used, your program may test the doubleword pointed to by register 1 for a delimiter made up of 1's in all of the bit positions.

Calling a CMS Command from a Program

You can call a CMS command from an assembler language program by setting up R1 to point to a tokenized parameter list and then issuing an SVC 202. For example:

```
PUNCHER  DS     0D
         DC     CL8'PUNCH'
         DC     CL8'NAME'
         DC     CL8'TYPE'
         DC     CL8'*'
         DC     CL8'('
         DC     CL8'NOH'
         DC     8X'FF'
```

Note: If you are using EXEC 2, refer to the *VM/SP EXEC 2 Reference* for information on parameter lists for EXEC 2 applications.

If your program is executing in the user area, you must be careful not to call any CMS command which also runs in the user area. The CMS commands that execute in the user area are identified in a table under "CMS Command Execution Characteristics" on page 3-19. Refer also to "Executing Program Modules" on page 8-7 for further considerations.

In your program, when you want to execute this command, you should load the address of the list into register 1, and issue the supervisor call instruction (SVC) as follows:

```
LA      1,PUNCHER
SVC 202
DC      AL4(ERROR)
. . .
```

When you issue an SVC 202, you must supply an error return address in the four bytes immediately after the SVC instruction. If the return code (register 15) contains a nonzero value after returning from the SVC call, control passes to the address specified unless the address is equal to 1. If the address is 1, return is made to the next instruction after the "DC AL4(1)" instruction. In the above example, control would go to the instruction at the label ERROR.

If you want to ignore errors, you can use the sequence:

```
LA      1,PUNCHER
SVC 202
DC      AL4(1)
```

If you do not specify an error address, control is returned to the next instruction after a normal return, but if there was an error executing the CMS command, your program terminates execution.

If you want to execute a CP command or an EXEC procedure from a program, you must use the CP and EXEC commands; for example:

```
SPOOL   DS      0D
        DC      CL8'CP'
        DC      CL8'SPOOL'
        DC      CL8'PRINTER'
        DC      CL8'CLASS'
        DC      CL8'S'
        DC      8X'FF'
EXEC    DC      CL8'EXEC'
        DC      CL8'PFSET'
        DC      8X'FF'
```

It is not possible to enter a parameter that is longer than eight characters this way.

As an alternative, you can use the CMS LINEDIT macro to call a CP command from a program. Specify DISP=CPCOMM on the macro instruction; for example:

```
LINEDIT TEXT='SPOOL E CLASS S',DISP=CPCOMM,DOT=NO
```

On return from the execution of the LINEDIT macro instruction, register 15 contains the return code from the CP command. The LINEDIT macro is described in *VM/SP CMS Command and Macro Reference.*

Another way to execute a CP command from a program is to use the DIAGNOSE x'08' instruction. For additional information on this, see the *VM/SP System Programmer's Guide.*

Creating Immediate commands

In addition to the CMS built-in Immediate commands, CMS provides facilities for you to create your own Immediate commands. Rules for creating your own Immediate commands are as follows:

1. Immediate commands can be created in three ways:

   a. Immediate commands can be creating from Assembler Language programs by issuing the IMMCMD macro. This macro associates a user-defined Immediate command name with the address of a user-supplied exit routine that receives control when the Immediate command is issued. Established Immediate commands can also be explicitly cancelled by the IMMCMD macro. If not explicitly cancelled, all Immediate commands created by the IMMCMD macro are automatically cancelled either upon return to the CMS command environment (if not in CMS SUBSET mode) or by entry to CMS abend.

   b. Immediate commands can be created from EXECs by use of the IMMCMD command. This command establishes and cancels Immediate commands and determines the status of the Immediate command. All Immediate commands not explicitly cancelled by the IMMCMD command are automatically cancelled either upon return to the CMS command environment (if not in CMS SUBSET mode) or by entry to CMS abend. User exit routines cannot be used with Immediate commands established by the IMMCMD command.

   c. Immediate commands can be created by using the immediate attribute that is supported by the NUCEXT function and the NUCXLOAD command. When a nucleus extension is declared with the immediate attribute, that nucleus extension is established as an Immediate command. By allowing nucleus extensions to be declared as Immediate commands, the following additional flexibility is provided:

      1) Immediate command routines can be created in free storage.

      2) Immediate commands can be permanently established for the duration of a CMS IPL (that is, they are not cleared during CMS end-of-command processing).

      3) Immediate commands can be invoked as exits during abend (SERVICE attribute) and end-of-command (ENDCMD attribute) processing.

      4) Immediate commands can be established to survive CMS abend (SYSTEM attribute).

      Nucleus extensions established as Immediate commands can be invoked as Immediate commands or as part of normal SVC 202 processing. When a nucleus extension is called as an Immediate command, the high-order byte of register 1 is set to X'06'.

      The immediate attribute is supported by the NUCEXT function (DECLARE, QUERY, CANCEL) and by the NUCXLOAD, NUCXDROP, and NUCXMAP commands.

2. Immediate commands can be 1 to 8 characters in length. Synonyms can be set up for Immediate commands just like they can be for regular CMS commands. Immediate commands or their synonyms must begin with a non-blank character.

3. Immediate commands are delimited by a blank. Any data following the blank is passed to the Immediate command routine as parameters. The capability to pass parameters is not applicable to Immediate commands declared by the IMMCMD command. Immediate commands and their parameters are subject to translation just as regular CMS commands are.

4. Immediate commands can be set up to override built-in CMS Immediate commands (for example, HX). However, built-in CMS commands cannot be cleared.

5. Immediate commands with the same name can override each other in a stack-like manner, with the most recent one declared being the one in effect.

6. The logical line end character is ignored on Immediate command input lines.

7. Both the IMMCMD macro, the NUCEXT function, and the NUCXLOAD command provide the capability to give control to an "exit" routine whenever a specific Immediate command is invoked. These exit routines receive control as an extension of CMS I/O interrupt handing. Therefore, they receive control with a PSW key of 0 and are disabled for interrupts. The exit routine must not perform any I/O operations or issue any SVCs that result in I/O operations. In addition, the exit routine must not enable itself for interrupts. DIAGNOSE instructions can be used within the exit, but the exit routine must not enable itself for interruptions that may be caused by the DIAGNOSE (for example, DIAGNOSE X'58').

8. Terminal users may optionally require that all Immediate commands be prefixed with an escape character. Use the SET IMESCAPE command to set the escape character. The status of IMESCAPE function can be determined by the QUERY command. For more details, see the *VM/SP CMS Command and Macro Reference.*

Executing Program Modules

MODULE files, in CMS, are nonrelocatable programs. Using the GENMOD command, you can create a module from any program that uses OS or CMS macros. When you create a module, it is generated at the virtual storage address at which it is loaded, for example:

```
load myprog
genmod testit
```

The CMS disk file, TESTIT MODULE A, that is created as a result of this GENMOD command, always begins execution at location X'20000', the beginning of the user program area.

If you want to call your own or CMS program modules using SVC 202 instructions, you must be careful not to execute a module that uses the same area of storage that your program occupies. If you want to call a module that executes at location X'20000', you can load the calling program at a higher location; for example:

```
load myprog (origin 30000
```

As long as the MODULE file called by MYPROG is no longer than X'10000' bytes, it will not overlay your program. Alternatively, either the calling or the called MODULE may be loaded as a nucleus extension. Refer to the CMS NUCXLOAD command in the *VM/SP CMS Command and Macro Reference* for more information on nucleus extensions.

Note: Many CMS disk-resident command modules execute in the user program area. This means that if you call a CMS command that runs in the user program area, you must be certain that it will not overlay your own program. Some CMS command modules issue the STRINIT macro or were created using the STR option of the GENMOD command.

Both cause the user area storage pointers to be reset. The reset condition may cause errors upon return to the original program (for example, when OS GETMAIN/FREEMAIN macros are issued in the user program).

The CMS commands that execute in the user program area or that reset the user area storage pointers are identified in a table under the heading "CMS Command Execution Characteristics" on page 3-19.

The Transient Program Area

To avoid overlaying programs executing in the user program area, you can generate program modules to run in the CMS transient area, which is a two-page area of storage that is reserved for the execution of programs that are called for execution frequently. Many CMS commands run in this area, which is located at X'E000'. Programs that execute in this area run disabled.

To generate a module to run in the transient area, use the ORIGIN TRANS option when you load the TEXT file into storage, then issue the GENMOD command:

```
load myprog (origin trans
genmod setup (str
```

Note: If a program running in the user area calls a transient routine in which a module was generated using the GENMOD command with the STR option, the user area storage pointers will be reset. This reset condition could cause errors upon return to the original program (for example, when OS GETMAIN/FREEMAIN macros are issued in the user program).

The two restrictions placed on command modules executing in the transient area are:

1. They may have a maximum size of 8192 bytes, since that is the size of the transient area. This size includes any free storage acquired by GETMAIN macros.

2. They must be serially reusable. When a program is called by an SVC 202, if it has already been loaded into the transient area, it is not reloaded.

The CMS commands that execute in the transient area are identified in a table under the heading "CMS Command Execution Characteristics" on page 3-19.

CMS Macro Instructions

There are a number of assembler language macros distributed with the CMS system that you can use when you are writing programs to execute in the CMS environment. These macros are in the macro libraries CMSLIB MACLIB and DMSSP MACLIB, which are normally located on the system disk.

- CMSLIB MACLIB contains macros from VM/370
- DMSSP MACLIB contains macros that are new or changed in VM/SP.

Note: When assembling programs that use CMS macros, both of these libraries should be identified via the GLOBAL command. DMSSP should precede CMSLIB in the search order.

There are macros to manipulate CMS disk files, to handle terminal communications, to manipulate unit record and tape input/output, and to trap interruptions. These macros are discussed in general terms here; for complete format descriptions, see the *VM/SP CMS Command and Macro Reference*.

Macros for Disk File Manipulation

Disk files are described in CMS by means of a file system control block (FSCB). The CMS macro instructions that manipulate disk files use FSCBs to identify and describe the files. When you want to manipulate a CMS file, you can refer to the file either by its file identifier, specifying 'filename filetype filemode' in quotation marks, or you can refer to the FSCB for the file, specifying FSCB=fscb, where fscb is the label on an FSCB macro.

To establish an FSCB for a file, you can use the FSCB macro instruction specifying a file identifier; for example:

```
INFILE   FSCB  'INPUT TEST A1'
```

You can also provide, on the FSCB macro instruction, descriptive information to be used by the input and output macros. If you do not code an FSCB macro instruction for a file, an FSCB is created in-line (following the macro instruction) when you code an FSREAD, FSWRITE, or FSOPEN macro instruction.

The format of an FSCB is listed in Figure 8-2, followed by a description of each of the fields.

Label			Description
FSCBCOMM	DC	CL8' '	File system command
FSCBFN	DC	CL8' '	Filename
FSCBFT	DC	CL8' '	Filetype
FSCBFM	DC	CL2' '	Filemode
FSCBITNO	DC	H'0'	Relative record number (RECNO)
FSCBBUFF	DC	A'0'	Address of buffer (BUFFER)
FSCBSIZE	DC	F'0'	Number of bytes to read or write (BSIZE)
FSCBFV	DC	CL2'F'	Record format - F or V (RECFM)

Figure 8-2 (Part 1 of 2). FSCB Format

Label			Description
FSCBFLG	EQU	FSCBFV+1	Flag byte
FSCBNOIT	DC	H'1'	Number of records to read or write (NOREC)
FSCBNORD	DC	AL4(0)	Number of bytes actually read
FSCBAITN	DC	AL4(0)	Extended FSCB relative record number
FSCBANIT	DC	AL4(1)	Extended FSCB relative number of records
FSCBWPTR	DC	AL4(0)	Extended FSCB relative write pointer
FSCBRPTR	DC	AL4(0)	Extended FSCB relative read pointer

Figure 8-2 (Part 2 of 2). FSCB Format

The fields FSCBAITN, FSCBANIT, FSCBWPTR, and FSCBRPTR are only generated in the FSCB when the extended format FSCB is requested (FORM=E is coded on the FSCB macro instruction). In this case, the fields FSCBITNO and FSCBNOIT are reserved fields. Extended format FSCBs must be used to manipulate files larger than 65,533 items. The labels shown above are not generated by the FSCB macro; to reference fields within the FSCB by these labels, you must use the FSCBD macro instruction to generate a DSECT.

FSCBCOMM: When the FSCBFN, FSCBFT, and FSCBFM fields are filled in, you can fill in the FSCBCOMM field with the name of a CMS command and use the FSCB as a parameter list for an SVC 202 instruction. (You must place a delimiter to mark the end of the command line.)

FSCBFN, FSCBFT, FSCBFM: The filename, filetype and filemode fields identify the CMS file to be read or written. You can code the fileid on a macro line in the format 'filename filetype filemode' or you can use register notation. If you use register notation, the register that you specify must point to an 18-byte field in the format:

```
FILEID   DC    CL8'filename'
         DC    CL8'filetype'
         DC    CL2'fm'
```

The fileid must be specified either in the FSCB for a file or on the FSREAD, FSWRITE, FSOPEN, or FSERASE macro instruction you use that references the file.

FSCBITNO: For an FSCB without the FORM=E option, the record or item number indicates the relative record number of the next record to be read or written; it can be changed with the RECNO option. The default value for this field is 0. When you are reading files, a 0 indicates that records are to be read sequentially, beginning with the first record in the file. When you are writing files, a 0 indicates that records are to be written sequentially, beginning at the first record following the end of the file, if the file already exists, or with record 1, if it is a new file.

For an FSCB generated with the FORM=E option, the FSCBAITN field contains the record or item number. The FSCBITNO field is reserved.

Whenever you read discontiguous files in CMS (that is, files with missing records), the input buffer will be filled with the appropriate number of bytes. Be aware that the flag byte in the FSCB may not reflect whether the input buffer contains generated data items from RDBUF.

FSCBBUFF: The buffer address, specified in the BUFFER option, indicates the label of the buffer from which the record is to be written or into which the record is to be read. You should always supply a buffer large enough to accommodate the longest record you expect to read or write. This field must be specified, either in the FSCB, or on the FSREAD or FSWRITE macro instruction.

FSCBSIZE: This field indicates the number of bytes that are read or written with each read or write operation. The default value is 0. If the buffer that you use represents the full length of the records you are going to be reading or writing, you can use the BSIZE option to set this field equal to your buffer length; when you are writing variable-length records, use the BSIZE operand to indicate the length of each record you write. This field must be specified.

FSCBFV: This two-character field indicates the record format (RECFM) of the file. The default value is F (fixed).

FSCBFLG The flag byte is X'20' indicating an extended FSCB generated when the FORM=E option is coded on the FSCB macro instruction.

FSCBNOIT: For an FSCB without the FORM=E option, this field contains the number of whole records that are to be read or written in each read or write operation. You can use the NOREC option with the BSIZE option to block and deblock records.

For an FSCB generated with the FORM=E option, the FSCBANIT field contains the number of whole records to be read or written. The FSCBNOIT field is reserved.

FSCBNORD Following a read operation, this field contains the number of bytes that were actually read, so that if you are reading a variable-length file, you can determine the size of the last record read. The FSREAD macro instruction places the information from this field into register 0.

FSCBAITN: The alternate record or item number indicates the relative record number of the next record to be read or written in an extended FSCB format. See the description of the FSCBITNO field for the usage of this field.

FSCBANIT: This field contains the alternate number of whole records in an extended FSCB format. See the description of the FSCBNOIT field for the usage of this field.

FSCBWPTR: The FSPOINT macro instruction uses this field to contain the alternate write pointer for an extended FSCB during a POINT operation.

FSCBRPTR: The FSPOINT macro instruction uses this field to contain the alternate read pointer for an extended FSCB during a POINT operation.

Using the FSCB

The following example shows how you might code an FSCB macro instruction to define various file and buffer characteristics, and then use the same FSCB to refer to different files:

```
          FSREAD 'INPUT FILE A1',FSCB=COMMON,FORM=E
          FSWRITE 'OUTPUT FILE A1',FSCB=COMMON,FORM=E
          .
          .
          .
COMMON    FSCB   BUFFER=SHARE,RECFM=V,BSIZE=200,FORM=E
SHARE     DS     CL200
```

In the above example, the fileid specifications on the FSREAD and FSWRITE macro instructions modify the FSCB at the label COMMON each time a read or write operation is performed. You can also modify an FSCB directly by referring to fields by a displacement off the beginning of the FSCB; for example:

```
MVC    FSCB+8,=CL8'NEWNAME'
```

moves the name NEWNAME into the filename field of the FSCB at the label FSCBFN.

As an alternative, you can use the FSCBD macro instruction to generate a DSECT and refer to the labels in the DSECT to modify the FSCB; for example:

```
          LA     R5,INFSCB
          USING  FSCBD,R5
          .
          .
          MVC    FSCBFN,NEWNAME
          .
          .
INFSCB    FSCB   'INPUT TEST A1',FORM=E
NEWNAME   DC     CL8'OUTPUT'
          FSCBD
```

In the above example, the MVC instruction places the filename OUTPUT into the FSCBFN (filename) field of the FSCB. The next time this FSCB is referenced, the file OUTPUT TEST is the file that is manipulated.

Reading and Writing CMS Disk Files

CMS disk files are sequential files; when you use CMS macros to read and write these files, you can access them sequentially with the FSREAD and FSWRITE macros. However, you may also refer to records in a CMS file by their relative record numbers, so you can, in effect, access records using a direct access method.

If you know which record you want to read or write, you can specify the RECNO option on the FSCB macro instruction, or on the FSOPEN, FSREAD, or FSWRITE macro instructions. When you use the RECNO option on the FSCB macro instruction, you must specify it as a self-defining term; for the FSOPEN, FSREAD, or FSWRITE macro instructions, you may specify either a self-defining term, as:

```
WRITE     FSWRITE FSCB=WFSCB,RECNO=10,FORM=E
```

or using register notation, as follows:

```
WRITE     FSWRITE FSCB=WFSCB,RECNO=(5),FORM=E
```

where register 5 contains the record number of the record to be read.

When you want to access files sequentially, the FSCBITNO field of the FSCB must be 0 for an FSCB without the FORM=E option; for an extended FSCB, the FSCBAITN field must be 0. This is the default value. When you are reading files

with the FSREAD macro instruction, reading begins with record number 1. When you are writing records to an existing file with the FSWRITE macro, writing begins following the last record in the file.

To begin reading or writing files sequentially beginning at a specific record number, you must specify the RECNO option twice: once to specify the relative record number at which you want to begin reading, and a second time to specify RECNO=0 so that reading or writing will continue sequentially beginning after the record just read or written. You can specify the RECNO option on the FSREAD or FSWRITE macro instruction, or you may change the FSCBITNO or FSCBAITN field in the FSCB for the file, as necessary for the FSCB form.

For example, to read the first record and then the 50th record of a file, you could code the following:

```
READ1     FSREAD FSCB=RFSCB,FORM=E
          FSWRITE FSCB=WFSCB,FORM=E
          LA     5,RFSCB
          USING FSCBD,5
          MVC    FSCBAITN,=F'50'
READ50    FSREAD FSCB=RFSCB,FORM=E
          FSWRITE FSCB=WFSCB,FORM=E
          .
          .
          .
RFSCB     FSCB 'INPUT FILE A1',BUFFER=COMMON,BSIZE=120,FORM=E
WFSCB     FSCB 'OUTPUT FILE A1',BUFFER=COMMON,BSIZE=120,FORM=E
COMMON    DS     CL120
          .
          .
          .
          FSCBD
```

In this example, the statements at the label READ1 write record 1 from the file INPUT FILE A1 to the file OUTPUT FILE A1. Then, using the DSECT generated by the FSCBD macro, the FSCBITNO field is changed because an extended FSCB is being used

FSCBAITN field is changed because an extended FSCB is being used and record 50 is read from the input file and written into the output file.

The "update-in-place" facility allows you to write blocks back to their previous location on disk. The "update-in-place" attribute of a CMS file is indicated by the filemode number 6.

Reading and Writing Variable-length Records:

When you read or write variable-length records, you must specify RECFM=V either in the FSCB for the file or on the FSWRITE or FSREAD macro instruction. The read/write buffer should be large enough to accommodate the largest record you are going to read or write.

To write variable-length records, use the BSIZE= option on the FSWRITE macro instruction to indicate the record length for each record you write. When you read variable-length records, register 0 contains, on return from FSREAD, the length of the record read.

The following example shows how you could read and write a variable-length file:

```
READ     FSREAD 'DATA CHECK A1',BUFFER=SHARE,BSIZE=130,ERROR=OUT,
           FORM=E
         FSWRITE 'COPY DATA A1',BUFFER=SHARE,BSIZE=(0),FORM=E
         B     READ
```

When you update files of variable-length records, the replacement record must be the same length as the original record. An attempt to write a record shorter or longer than the original record results in truncation of the file at the specified record number. No error return code is given.

End-of-File Checking

You can specify the ERROR= operand with the FSREAD or FSWRITE macro instruction, so that an error handling routine receives control in case of an error. In CMS, when an end of file occurs during a read request, it is treated as an error condition. The return code is always 12. If you specify an error handling routine on the FSREAD macro instruction, then the first thing this routine can do is check for a 12 in register 15.

Your error handling routine may also check for other types of errors. See the macro description in the *VM/SP CMS Command and Macro Reference* for details on the possible errors and the associated return codes.

Opening and Closing Files

Usually, CMS opens a file whenever an FSREAD or FSWRITE macro instruction is issued for the file. When control returns to CMS from a calling program, all files accidentally left open are closed by CMS, so you do not have to close files at the end of a program.

For a minidisk in 512-, 1K-, 2K-, or 4K-byte block format, a file may be open for concurrent read and write operations, and an FSCLOSE need not be issued when switching from reading to writing, or vice versa. For example:

```
         LA    3,2

READ     FSREAD FSCB=UPDATE,RECNO=(3),ERROR=READERR,FORM=E
         .
         .
         .
         FSWRITE FSCB=UPDATE,RECNO=(3),ERROR=WRITERR,FORM=E
         LA    3,1(3)
         B     READ
         .
         .
         .
UPDATE   FSCB  'UPDATE FILE A1',BUFFER=BUF1,BSIZE=80,FORM=E
```

When you are running long running applications or running disconnected, include several FSCLOSE macros to each file referenced. This insures that changes to the file are reflected on the disk in the event that the user is forced off the system. This consideration is important when running on 512-, 1K-, 2K-, or 4K-byte block disks since the disk directory is not updated until all of the files on the disk are closed.

If you want to read and write records from the same file on an 800-byte block format minidisk, you must issue an FSCLOSE macro instruction to close the file whenever you switch from reading to writing. For example:

```
          LA    3,2
READ      FSREAD FSCB=UPDATE,RECNO=(3),ERROR=READERR
          FSCLOSE FSCB=UPDATE
          .
          .
          .
          FSWRITE FSCB=UPDATE,RECNO=(3),ERROR=WRITERR
          FSCLOSE FSCB=UPDATE
          LA    3,1(3)
          B     READ
          .
          .
          .
UPDATE    FSCB  'UPDATE FILE A1',BUFFER=BUF1,BSIZE=80
```

To execute a loop to read, update, and rewrite records, you must read a record, close the file, write a record, close the file, and so on. Since closing a file repositions the read pointer to the beginning of the file and the write pointer at the end of the file, you must specify the relative record number (RECNO) for each read and write operation. In the above example, register 3 is used to contain the relative record number. It is initialized to begin reading with the second record in the file and is increased by one following each write operation.

When you use an EXEC to execute a program to read or write a file, the file is not closed by CMS until the EXEC completes execution. Therefore, if you read or write the same file more than once during the EXEC procedure, you must use an FSCLOSE macro instruction to close the file after using it in each program, or use the FSOPEN macro instruction to open it before each use. Otherwise, the read or write pointer is positioned as it was when the previous program completed execution.

Creating New Files:

When you want to begin writing a new file using CMS data management macros, there are two ways to ensure that the file you want to create does not already exist. One way is to issue the FSSTATE macro instruction to verify the existence of the file.

A second way to ensure that a file does not already exist is to issue an FSERASE macro instruction to erase the file. If the file does not exist, register 15 returns with a code of 28. If the file does exist, it is erased. See Figure 8-3 on page 8-16 for an illustration of a sample program using CMS data management macros.

```
LINE    SOURCE STATEMENT

BEGIN   CSECT                                          (1)
        PRINT NOGEN
        USING *,12          ESTABLISH ADDRESSABILITY
        LR    12,15
        ST    14,SAVE
        LA    2,8(,1) R2=ADDR OF INPUT FILEID IN PLIST     (2)
        LA    3,32(,1) R3=ADDR OF OUTPUT FILEID IN PLIST
* DETERMINE IF INPUT FILE EXISTS
        FSSTATE (2),ERROR=ERR1,FORM=E
*
* READ A RECORD FROM INPUT FILE AND WRITE ON OUTPUT FILE
RD      FSREAD (2),ERROR=EOF,BUFFER=BUFF1,BSIZE=80,FORM=E    (3)
        FSWRITE (3),ERROR=ERR2,BUFFER=BUFF1,BSIZE=80,FORM=E
        B     RD            LOOP BACK FOR NEXT RECORD
*
* COME HERE IF ERROR READING INPUT FILE
EOF     C     15,=F'12'     END OF FILE ?    (4)
        BNE   ERR3          ERROR IF NOT
        LA    15,0          ALL O.K. - ZERO OUT R15
        B     EXIT          GO EXIT
* IF INPUT FILE DOES NOT EXIST
ERR1    WRTERM 'FILE NOT FOUND',EDIT=YES
        B     EXIT
*
* IF ERROR WRITING FILE
ERR2    LR    10,15         SAVE RET CODE IN REG 10      (5)
        LINEDIT TEXT='ERROR CODE .... IN WRITING FILE',SUB=(DEC,(10))
        B     EXIT
*
* IF READING ERROR WAS NOT NORMAL END OF FILE
ERR3    LR    10,15         SAVE RET CODE IN REG 10      (5)
        LINEDIT TEXT='ERROR CODE .... IN READING FILE',SUB=(DEC,(10))
*
EXIT    L     14,SAVE       LOAD RETURN ADDRESS
        BR    14            RETURN TO CALLER
*
BUFF1   DS    CL80
SAVE    DS    F
        END
```

Notes:

(1) The program might be invoked with a parameter list in the format progname INPUT FILE A1 OUTPUT FILE A1. This line is placed in a parameter list by CMS routines and addressed by register 1 (see note 2).

(2) The parameter list is a series of doublewords, each containing one of the words entered on the command line. Thus, 8 bytes past register 1 is the beginning of the input fileid; 24 bytes beyond that is the beginning of the second fileid.

(3) The FSREAD and FSWRITE macros cause the files to be opened; no open macro is necessary. CMS routines close all open files when a program completes execution (except CMS EXEC files).

(4) The return code in register 15 is tested for the value 12, which indicates an end-of-file condition. If it is the end of the file, the program exits; otherwise, it writes an error message.

(5) The dots in the LINEDIT macro are substituted, during execution, with the decimal value in register 10.

Figure 8-3. A Sample Listing of a Program that Uses CMS Macros

CMS Macros for Terminal Communications

There are four CMS macros you can use to write interactive, terminal-oriented programs. They are RDTERM, WRTERM, LINEDIT, and WAITT. RDTERM and WRTERM only require a read/write buffer for sending and receiving lines from the terminal. The third, LINEDIT, has a substitution and translation capability.

When you use the WRTERM macro to write a line to your terminal you can specify the actual text line in the macro instruction, for example:

```
DISPLAY  WRTERM 'GOOD MORNING'
```

You can also specify the message text by referring to a buffer that contains the message.

The RDTERM macro accepts a line from the terminal and reads it into a buffer you specify. You could use the RDTERM and WRTERM macros together, as follows:

```
WRITE     WRTERM 'ENTER LINE'
READ      RDTERM BUFFER
          LR   3,0
REWRITE   WRTERM BUFFER,(3)
            .
            .
            .
BUFFER    DS   CL130
```

In this example, the WRTERM macro results in a prompting message. Then the RDTERM macro accepts a line from the terminal and places it in the buffer BUFFER. The length of the line read, contained in register 0 on return from the RDTERM macro, is saved in register 3. When you specify a buffer address on the WRTERM macro instruction, you must specify the length of the line to be written. Here, register notation is used to indicate that the length is contained in register 3.

The LINEDIT macro converts decimal and hexadecimal data into EBCDIC, and places the converted value into a specified field in an output line. There are list and execute forms of the macro instruction, which you can use in writing reentrant code. Another option allows you to write lines to the offline printer. The LINEDIT macro is described, with examples, in *VM/SP CMS Command and Macro Reference*. Figure 8-3 on page 8-16 shows how you might use the LINEDIT macro to convert and display CMS return codes.

The WAITT (wait terminal) macro instruction can help you to synchronize input and output to the terminal. If you are executing a program that reads and writes to the terminal frequently, you may want to issue a WAITT macro instruction to halt execution of the program until all terminal I/O has completed.

CMS Macros for Unit Record and Tape I/O

CMS provides macros to simplify reading and punching cards (RDCARD and PUNCHC), and creating printer files (PRINTL). When you use either the PUNCHC or PRINTL macros to write or punch output files while a program is executing, you should remember to issue a CLOSE command for your virtual printer or punch when you are finished. You can do this either after your program returns control to CMS, by entering:

```
cp close e

-- or --

cp close d
```

or, you can set up a parameter list with the command line CP CLOSE E or CP CLOSE D and issue an SVC 202.

The tape control macros, RDTAPE, WRTAPE and TAPECTL, can read and write CMS files from tape, or control the positioning of a tape.

Interruption Handling Macros

You can set up routines in your programs to handle interruptions caused by I/O devices, by SVCs, or by external interruptions using the HNDINT, HNDSVC, or HNDEXT macro instructions.

With the HNDINT macro instruction, you can specify addresses that are to receive control when an interruption occurs for a specified device. If the WAIT option is used for a device specified in the HNDINT macro instruction, then the interruption handling routine specified for the device does not receive control until after the WAITD macro instruction is issued for the device.

You can use the HNDSVC macro instruction to trap supervisor call instructions of particular numbers, if, for example, you want to perform some additional function before passing control or you do not want any SVCs of the specified number to be executed.

The CP EXTERNAL command simulates external interruptions in your virtual machine; if you want to be able to pass control to a particular internal routine in the event of an external interruption, you can use the HNDEXT macro instruction.

Updating Source Programs Using CMS

As you test and modify programs, you may want to keep backup copies of the source programs. Then you can always return to a certain level of a program in case you have an error. CMS provides several approaches to the problem of program backup: the method you choose depends on the complexity of your project, the changes you want to make, and the size of your programs.

The simplest method is to make a copy of the current source file under a new name. You can do this using either the COPYFILE command or the editor. If you use the COPYFILE command, your command line might be:

```
copyfile account assemble a oldacct assemble a
```

Then, you can use the editor to modify ACCOUNT ASSEMBLE; the file OLDACCT ASSEMBLE contains your original source file.

You can make a copy of your source file using the editor directly. For example, if you issue:

```
xedit account assemble
set fname newacct
```

then any subsequent changes you make to the file ACCOUNT ASSEMBLE are written into the file NEWACCT ASSEMBLE. When you issue a FILE or SAVE subcommand, your source file remains intact.

After your changes to the source program have been tested, you can replace the source file with your new copy.

The UPDATE Philosophy

While the procedures outlined above for modifying programs are suitable for many applications, they may not be adequate in a situation where several programmers are applying changes to the same source code. These procedures also have the drawback of not providing you with a record of what has been changed. After using the editor, you do not have a record of the lines that have been deleted, added, replaced, and so on, unless you manually add comments to the code, insert special characters in the serialization column, or use some technique that records program activity.

The UPDATE command and the XEDIT UPDATE option provide a way for you to modify a source program without affecting the original. UPDATE produces an update log, indicating the changes that have been made; both UPDATE and XEDIT have the capability of combining multiple updates at one time, so that changes made by different programmers or changes made at different times can be combined into a single output file.

The UPDATE command and the XEDIT UPDATE option are the basic elements of the entire VM/SP updating scheme and are used by system programmers who maintain VM/SP at your installation. Although the input filetypes used by the UPDATE command default to ASSEMBLE file characteristics, neither the UPDATE command nor the XEDIT UPDATE option is limited to assembler language programs, nor is it limited to system programming applications. You can use it to modify and update any fixed-length, 80-character file that does not have data in columns 73 through 80.

Update Files

A simple update involves two input files:

- The source file, which is the program you want to update.
- An update file, usually created by XEDIT, containing control statements that describe the changes you want to make.

The control statement file usually has a filetype of UPDATE. For convenience, you can give it the same filename as your source file. For example, if you want to update the file SAMPLE ASSEMBLE, you would create a file named SAMPLE UPDATE using the XEDIT UPDATE option. To apply the changes in the update file, you issue the command:

```
update sample
```

The default values used by the UPDATE command are filetypes of ASSEMBLE and UPDATE for the source and update files, respectively. If you are updating a COBOL source program named READY COBOL with an update file named UPDATE READY, you would issue the command:

```
update ready cobol a update ready a
```

After an UPDATE command completes processing, the input files are not changed; two new files are created. One of them contains the updated source file, with a filename that is the same as the original source file but preceded by a dollar sign ($). Another file, containing a record of updates is also created; it has a filename that is the same as the source file and a filetype of UPDLOG. For example:

Source Files	Output Files
SAMPLE ASSEMBLE	$SAMPLE ASSEMBLE
SAMPLE UPDATE	SAMPLE UPDLOG
READY COBOL	$READY COBOL
UPDATE READY	READY UPDLOG

Now, you can assemble or compile the new source file created by the UPDATE command.

Creating an Update File

An update file can be created easily using the XEDIT UPDATE option. Using XEDIT, there is no need for the programmer to enter the control statements in the UPDATE file. These are generated automatically by the editor. For example:

```
xedit ready cobol a (upd
```

specifies that a file called READY COBOL is to be edited and all updates to the file are placed in a separate file called READY UPDATE along with the appropriate control statements

The XEDIT UPDATE option expects source files to have sequence numbers in columns 73 through 80. Before you can create an UPDATE file you must use the XEDIT SERIAL subcommand to sequence your files. To generate these sequence numbers, you should issue:

```
serial all
```

prior to issuing a FILE or SAVE subcommand when you are editing a file. Alternately, you can preface sequence numbers with a three character identifier, usually the first three characters of the fileneme. If you issue:

```
serial on
```

XEDIT will write sequence numbers in columns 76 through 80 of your file. Columns 73 through 75 will contain the first three characters of the filename. If SERIAL ON is specified, you must also specify the NOSEQ8 option on the XEDIT command to tell the editor to expect a sequence of numbers only in columns 75 through 80. For example:

```
xedit ready cobol a (upd noseq8
```

Using XEDIT with an Existing Update File

If an update file already exists for a given source file and you wish to either (1) to browse the source file with the updates applied or (2) to continue updating the source file, you issue the same XEDIT command that you entered when you created the update file. For example:

```
xedit ready cobol a (upd
```

will apply all updates contained in READY UPDATE to the source file READY COBOL and display the resulting file on the screen. Any updates created during this editing session will be added to to those already contained in READY UPDATE. Again, all control statements will automatically be generated by XEDIT. More information about the XEDIT UPDATE option can be found in the *VM/SP CMS Command and Macro Reference.*

UPDATE Control Statements

The control statements used by the UPDATE command are similar to those used by the OS IEBUPDTE utility program or the DOS MAINT program UPDATE function.

Each UPDATE statement must have the characters ./ in columns one and two, followed by one or more blanks. The statements are described below, with examples.

SEQUENCE Statement: This statement tells the UPDATE command that you want to number or renumber the records in a file. Sequence numbers are written in columns 73 through 80. For example, the statement:

```
./ S  1000
```

indicates that you want sequence numbering to be done, in increments of 1000, with the first statement numbered 1000. The SEQUENCE statement is convenient if you want to apply updates to a file that does not already have sequence numbers. In this case, you may want to use the REP (replace) option of the UPDATE command, so that instead of creating a new file ($filename), the original source file is replaced:

```
update sample (rep
```

INSERT Statement: This statement precedes new records that you want to add to a source file. The INSERT statement tells the UPDATE command where to add the new records. For example, the lines:

```
./ I  1600
TEST2  TM   HOLIDAY,X'02'    HOLIDAY?
       BNO  VACATION         NOPE...VACATION
```

result in the two lines of code being inserted into the output file following the statement numbered 00001600. The inserted lines are flagged with asterisks in columns 73 through 80. The INSERT statement also allows you to request that new statements be sequenced; see "Sequencing Output Records."

DELETE Statement: This statement tells the UPDATE command which records you want to delete from the source file. If your UPDATE file contains:

```
./ D  2500
```

then only the record 00002500 is deleted. If the file contains

```
./ D  2500  2800
```

then all the statements from 2500 through 2800 are deleted from the source file.

REPLACE Statement: The REPLACE statement allows you to replace one or more records in the source file. It precedes the new records you want to add. It is a combination of the DELETE and INSERT statements. For example, the lines

```
./ R 38000 38500
PLIST    DS    0D
         DC    CL8'TYPE'
         DC    CL8' '
         DC    CL8'FILE'
         DC    CL8'A1'
         DC    8X'FF'
```

replace existing statements numbered 38000 through 38500 with the new lines of code. As with the INSERT statement, new lines are not automatically resequenced.

COMMENT Statement: Use this statement when you want to place comments in the update log file. For example, the line:

```
./ * Changes by John J. Programmer
```

is not processed by the UPDATE command when it creates the new source file, but it is written into the update log file.

Sequencing Output Records

The UPDATE command expects source files to have sequence numbers in columns 73 through 80. If you use the SERIAL subcommand of the CMS editor to sequence your files, the sequence numbers are usually written in columns 76 through 80; columns 73 through 75 contain a three-character identifier which is usually the first three characters of the filename. If you want an eight-character sequence number, you must use the subcommand:

```
serial all
```

prior to issuing a FILE or SAVE subcommand when you are editing the file. Or, you can create an UPDATE file with the single record:

```
./ S
```

and issue the UPDATE command to sequence the file.

If you use the UPDATE command with a file that has been sequenced using the CMS editor's default values, you must use the NOSEQ8 option. Otherwise, the UPDATE command cannot process your input file. The command:

```
update sample (noseq8
```

tells UPDATE to use only columns 76 through 80 when it looks for sequence numbers. Figure 8-4 shows the four files involved in a simple update, and their contents.

The Source File, SAMPLE ASSEMBLE

```
SAMPLE   CSECT                                                          00000100
         USING SAMPLE,R12                                               00000200
         LR    R12,R15                                                  00000300
         ST    R14,SAVRET                                               00000400
         LINEDIT TEXT='PLEASE ENTER YOUR NAME'                          00000500
         RDTERM NAME                                                    00000600
         LINEDIT TEXT='PLEASE ENTER YOUR AGE'                           00000700
         RDTERM AGE                                                     00000800
         LINEDIT TEXT='HI, ..........., YOU JUST TOLD ME YOU ARE .....',x00000900
               SUB=(CHARA,NAME,CHARA,AGE),RENT=NO                       00001000
         L     R14,SAVRET                                               00001100
         BR    R14                                                      00001200
         EJECT                                                          00001300
NAME     DC    CL130' '                                                 00001400
AGE      DC    CL130' '                                                 00001500
SAVRET   DC    F'0'                                                     00001600
         REGEQU                                                         00001700
         END                                                            00001800
```

The Update File, SAMPLE UPDATE

```
./ * REVISION BY DLC                                                    SAM00010
./ R 500                                                                SAM00020
         LINEDIT TEXT='WHAT''S YOUR NAME?',DOT=NO                       SAM00030
./ R 700 1000                                                           SAM00040
         LINEDIT TEXT='HI, ..........., ENTER THE DOCNAME',            xSAM00050
               SUB=(CHARA,NAME)                                         SAM00060
         RDTERM NAME                                                    SAM00070
         MVC   DOCFN,NAME                                               SAM00080
         LA    1,PLIST                                                  SAM00090
         SVC 202                                                        SAM00100
         DC    AL4(ERROR)                                               SAM00110
RETURN   EQU   *                                                        SAM00120
./ I 1200                                                               SAM00130
ERROR    EQU   *                                                        SAM00140
         WRTERM 'FILE NOT FOUND'                                        SAM00150
         B     RETURN                                                   SAM00160
./ D 1500                                                               SAM00170
./ I 1600                                                               SAM00180
PLIST    DS    0D                                                       SAM00190
         DC    CL8'TYPE'                                                SAM00200
DOCFN    DC    CL8' '                                                   SAM00210
         DC    CL8'FILE'                                                SAM00220
         DC    CL8'A1'                                                  SAM00230
         DC    8X'FF'                                                   SAM00240
```

Figure 8-4 (Part 1 of 2). Updating Source Files with the UPDATE Command

The Record of Updates File, SAMPLE UPDLOG

```
UPDATING 'SAMPLE   ASSEMBLE A1' WITH 'SAMPLE   UPDATE   A1'            UPDATE LOG -- PAGE      1
        ./ * REVISION BY DLC
        ./ R 500
DELETING...             LINEDIT TEXT='PLEASE ENTER YOUR NAME'                                  00000500
INSERTING...            LINEDIT TEXT='WHAT''S YOUR NAME?',DOT=NO                               ********
        ./ R 700 1000
DELETING...             LINEDIT TEXT='PLEASE ENTER YOUR AGE'                                   00000700
                        RDTERM AGE                                                             00000800
                        LINEDIT TEXT='HI, ..........., YOU JUST TOLD ME YOU ARE .....',x00000900
                              SUB=(CHARA,NAME,CHARA,AGE),RENT=NO                               00001000
INSERTING...            LINEDIT TEXT='HI, ..........., ENTER THE DOCNAME',                    x********
                              SUB=(CHARA,NAME)                                                 ********
                        RDTERM NAME                                                            ********
                        MVC   DOCFN,NAME                                                       ********
                        LA    1,PLIST                                                          ********
                        SVC 202                                                                ********
                        DC    AL4(ERROR)                                                       ********
                RETURN  EQU   *                                                                ********
        ./ I 1200
INSERTING...    ERROR   EQU   *                                                                ********
                        WRTERM 'FILE NOT FOUND'                                                ********
                        B     RETURN                                                           ********
        ./ D 1500
DELETING...     AGE     DC    CL130' '                                                         00001500
        ./ I 1600
INSERTING...    PLIST   DS    0D                                                               ********
                        DC    CL8'TYPE'                                                        ********
                DOCFN   DC    CL8' '                                                           ********
                        DC    CL8'FILE'                                                        ********
                        DC    CL8'A1'                                                          ********
                        DC    8X'FF'                                                           ********
```

The Updated Output File, $SAMPLE ASSEMBLE

```
SAMPLE  CSECT                                                                                  00000100
        USING SAMPLE,R12                                                                       00000200
        LR    R12,R15                                                                          00000300
        ST    R14,SAVRET                                                                       00000400
        LINEDIT TEXT='WHAT''S YOUR NAME?',DOT=NO                                               ********
        RDTERM NAME                                                                            00000600
        LINEDIT TEXT='HI, ..........., ENTER THE DOCNAME',                                    x********
              SUB=(CHARA,NAME)                                                                 ********
        RDTERM NAME                                                                            ********
        MVC   DOCFN,NAME                                                                       ********
        LA    1,PLIST                                                                          ********
        SVC 202                                                                                ********
        DC    AL4(ERROR)                                                                       ********
RETURN  EQU   *                                                                                ********
        L     R14,SAVRET                                                                       00001100
        BR    R14                                                                              00001200
ERROR   EQU   *                                                                                ********
        WRTERM 'FILE NOT FOUND'                                                                ********
        B     RETURN                                                                           ********
        EJECT                                                                                  00001300
NAME    DC    CL130' '                                                                         00001400
SAVRET  DC    F'0'                                                                             00001600
PLIST   DS    0D                                                                               ********
        DC    CL8'TYPE'                                                                        ********
DOCFN   DC    CL8' '                                                                           ********
        DC    CL8'FILE'                                                                        ********
        DC    CL8'A1'                                                                          ********
        DC    8X'FF'                                                                           ********
        REGEQU                                                                                 00001700
        END                                                                                    00001800
```

Figure 8-4 (Part 2 of 2). Updating Source Files with the UPDATE Command

The INSERT and REPLACE statements allow you to control the numbering increment of records that you add to a source file. Notice, in Figure 8-4 on page 8-23 that inserted records have the character string '********' in columns 73 through 80. If you want sequence numbers on the inserted records, you must do two things:

1. Use the INC option on the UPDATE command line. If you use the CTL option, you do not have to specify the INC option. The CTL option is described below, under "Multiple Updates."

2. Include a dollar sign ($) on the INSERT or REPLACE statement, optionally followed by operands indicating how the records should be sequenced.

For example, to sequence the records added in Figure 8-4 on page 8-23 the control statements would appear as:

```
./ R  500 $
./ R  700 1000 $
./ I  1200 $
./ I  1600 $
```

and you would issue the UPDATE command:

```
update sample (inc
```

The UPDATE command sequences inserted records by increments of 10. If you want to control the numbering, for example, if you need to insert more than 10 statements between two existing statements, you can specify an alternate sequencing scheme:

```
 ./ I  1800 $ 1805  5
.
.
.
```

Records introduced following this INSERT statement are numbered 00001805, 00001810, 00001815, and so on. (If the NOSEQ8 option is in effect, then the records would be XXX01805, XXX01810, and so on, where XXX is the three-character identifier used in columns 73 through 75.)

Multiple Updates

If you have several UPDATE files to apply to the same source, you may apply them in a series of UPDATE commands. For example, if you have updates named FICA UPDTUP1, FICA UPDTUP2, and FICA UPDTUP3 to apply to the source file FICA PLIOPT, you could do the following:

1. Update the source file with TEST1 UPDATE:

   ```
   update fica pliopt a fica updtup1
   ```

2. Update the source file produced by the above command with the TEST2 UPDATE:

   ```
   update $fica pliopt a fica updtup2
   ```

3. Update the new source file with TEST3:

   ```
   update $$fica pliopt a fica updtup3
   ```

This final UPDATE command produces the file $$$FICA PLIOPT, which is now the fully updated source file. This method is cumbersome, however, particularly if you have many updates to apply and they must be applied in a particular order. Therefore, the UPDATE command provides a multilevel update scheme, which you can use to apply many updates at one time, in a specified order.

To apply multilevel updates, you must have a control file, which by convention has a filetype of CNTRL and a filename that is the same as the source input file. Therefore, to apply the three update files to FICA PLIOPT, you should create a file named FICA CNTRL.

The Control File

A control file is actually a list: it does not contain any actual update control statements (INSERT, DELETE, and so on), but rather it indicates what update files should be applied, and in what order. In the case of a multilevel update, all the update files must have the same filename as the source file. Therefore, only the *filetypes* need be specified in the control file to uniquely identify the update file. In fact, if all your update files have filetypes beginning with the characters UPDT, you need only specify the unique part of the filetype. The control file for FICA PLIOPT, named FICA CNTRL, may typically look like the following:

```
TEXT MACS PLILIB
FICA3 UP3
FICA2 UP2
FICA1 UP1
```

The first record in the control file must be a MACS record. The second field in this record must be "MACS" and it may be followed by up to eight macro library names. Every record in the control file must have an "update level identifier"; in this example, the update level identifiers are TEXT on the MACS record, FICA1 for the UP1 record, and so on. The update level identifier may have a maximum of five characters. See the "STK option" for more details about the "update level identifier."

The UPDATE command only uses the MACS record and the update level identifier under special circumstances. These are described later, under "VMFASM EXEC Procedure." For now, you only need to know that these things must be in a control file in order for the UPDATE command to execute properly.

To update FICA PLIOPT, then, you would issue the UPDATE command as follows:

```
update fica pliopt (ctl
```

When you use the CTL option, and you do not specify the name of a control file, the UPDATE command looks for a control file with the filetype of CNTRL and a filename that is the same as the source file. From the control file, it reads the filetypes of the updates to be applied. In this example, it searches for the file FICA UPDTUP1 and if found, applies the updates; then UPDATE searches for FICA UPDTUP2, and applies those updates, if any. Last it searches for FICA UPDTUP3, and applies those updates.

Notice that the updates are applied from the bottom of the control file, toward the top. This becomes important when an update is dependent on a previous update. For example, if you add some lines to a file in FICA UPDTUP1, then modify one of those lines in FICA UPDTUP2, it is important that UPDTUP1 was applied first.

Alternate Ways of Specifying Multilevel Update Files:

The example above, showing FICA CNTRL and UPDTxxxx files, illustrates a naming scheme using the UPDATE command defaults. You can override the default filetypes for the control file's filename and filetype, as well as filetypes for the update files.

If you name a control file GROUPA CNTRL, for example, you can specify the name of the control file on the UPDATE command line:

```
update fica pliopt a groupa cntrl (ctl
```

Similarly, if your update files have unique filetypes, you must specify the entire filetype in the control file. If your updates to FICA PLIOPT are named FICA TEST1, FICA TEST2, and FICA TEST3, your control file may look like the following:

```
TEXT MACS PLILIB
FICA3 TEST3
FICA2 TEST2
FICA1 TEST1
```

Regardless of the filetypes you choose, however, the filenames must always be the same as the filename of the input source file.

AUX Files

The two levels of update processing shown so far may be adequate for your applications. There is, however, an additional level, or step, in the update structure that the VM/SP procedures use and which you may want to use also.

These techniques may be useful when you have more than one set of updates to apply to a source program. For example, you may have two groups of programmers who are working on different sets of changes for the same source file. Each group may create several update files, and have a unique control file. When you combine these changes, you could create one control file, or you can use what are known as auxiliary control files.

The updating structure for auxiliary control files is based on conventions for assigning filenames and filetypes. If a control file contains an entry that begins with the characters "AUX," the UPDATE command assumes that the file "fn AUXnnnn" contains a list of filetypes, not UPDATE control statements. For example, if the file SAMPLE ASSEMBLE is being updated with a control file that contains the record:

```
TEST1 AUXLIST
```

then SAMPLE AUXLIST does not contain UPDATE control statements; it contains entries indicating the *filetypes* of the update files, all of which must have the same filename, SAMPLE.

Let's expand the example to see how this structure works. We have the source file, SAMPLE ASSEMBLE. The file SAMPLE CNTRL contains the entries:

```
TEXT MACS
3676 AUXLIST
```

The file, SAMPLE AUXLIST may look like the following:

```
TEST1
FIXLOOP
BYPASS
```

The files:

```
SAMPLE TEST1
SAMPLE FIXLOOP
SAMPLE BYPASS
```

all contain UPDATE control statements (INSERT, DELETE, and so on) that are to be applied to the file SAMPLE ASSEMBLE. As with control file processing, the updates are applied from the bottom of the AUX file, so that the updates in SAMPLE BYPASS are applied first, then the updates in SAMPLE FIXLOOP, and so on. For an illustration of a set of update files, see Figure 8-5 on page 8-29.

Since the updating scheme uses only filetypes to uniquely identify update files, it is possible to use the same control file to update different source input files. For example, issue the following command when using the control file REPORT CNTRL shown in Figure 8-5 .

```
update fica pliopt a report cntrl (ctl
```

The UPDATE command begins searching for updates to apply to FICA PLIOPT, based on the entries in REPORT CNTRL: it searches for FICA AUXFIX, which may contain entries pointing to update files; then it searches for FICA UPDTREP1, and so on.

As long as all updates and auxiliary files associated with a source file have the same filename as the source file, the updates are uniquely identifiable, so the same control file can be used to update various source files. VM/SP takes advantage of this capability in its own updating procedures. By maintaining strict naming conventions, updates to various CP and CMS modules are easily controlled and identified.

A control file may point to many AUX files in addition to many UPDT files. You can modify a control file when you want to control which updates are applied to a program, or you may have several control files, and specify the name of the control file you want to use on the UPDATE command line. There is a lot of flexibility in the UPDATE command processing; you can implement procedures and conventions for your individual applications.

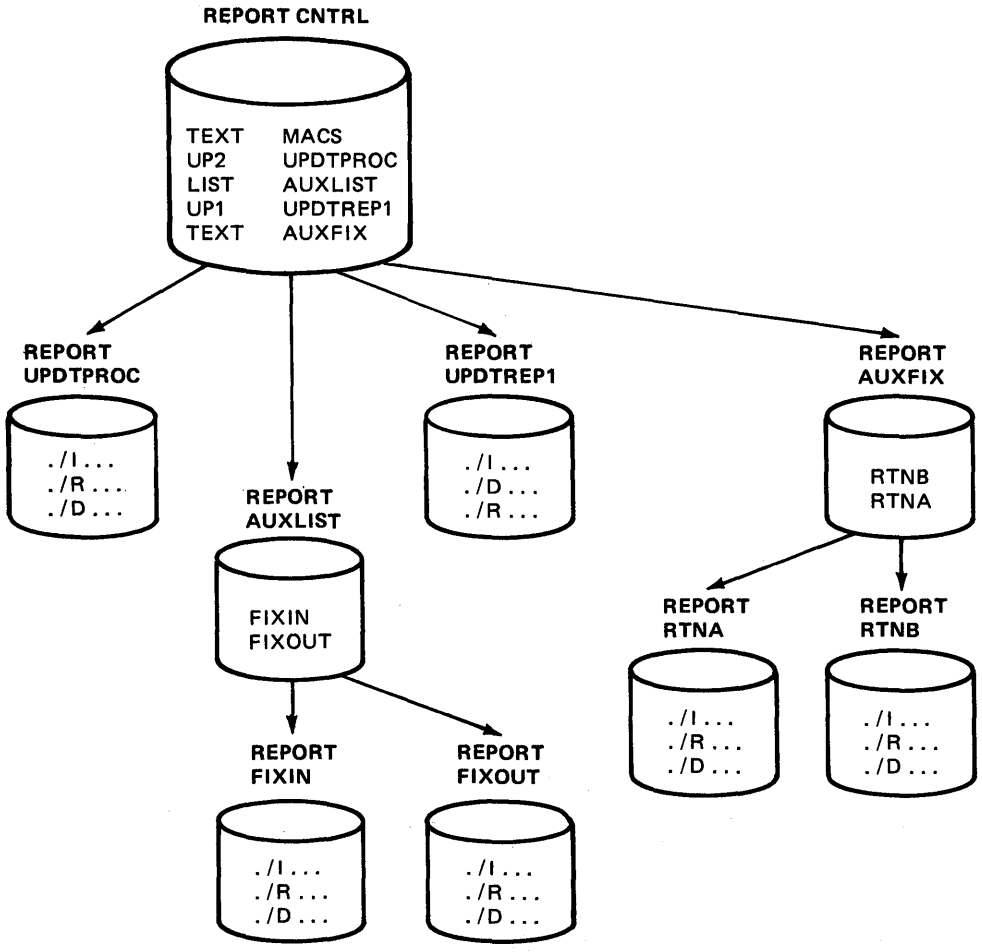
Multiple Updates with XEDIT

The XEDIT CTL option can be used to create multiple updates to a source file. First, create a control file listing the updates to be applied to a source file. Initially, you might have only the MACS record and one UPDATE filetype specified. For example, you can create a file called FICA CNTRL that contains:

```
TEXT MACS PLILIB
FICA1 UPDTUP1
```

Next, specify the control file name that you have created after the XEDIT CTL option. For example:

```
xedit fica pliopt (ctl fica
```




```
update report assemble a (ctl)
UPDATING 'REPORT ASSEMBLE A1' WITH 'REPORT RTNA A1'.
UPDATING WITH 'REPORT RTNB A1'.
UPDATING WITH 'REPORT UPDTREP1 A1'.
UPDATING WITH 'REPORT FIXOUT A1'.
UPDATING WITH 'REPORT FIXIN A1'.
UPDATING WITH 'REPORT UPDTPROC A1'.
R;
```

Figure 8-5. An Update with a Control File

The editor will search for an update file called FICA UPDTUP1 and apply all updates contained in this file. If the update file does not exist, XEDIT will create a file called FICA UPDTUP1 which will contain all changes made to the source file during the editing session in addition to the required control statements.

If you wish to add another level of updates to your source file, insert a new update filetype in your control file after the MACS record, for example:

```
TEXT MACS PLILIB
FICA2 UPDTUP2
FICA1 UPDTUP1
```

Then, XEDIT your source file again, specifying the CTL option, for example:

```
xedit fica pliopt (ctl fica
```

XEDIT applies all updates contained in FICA UPDTUP1 to the source file FICA PLIOPT. After the resulting file is displayed, any additional updates and the necessary control statements will automatically be inserted in another update file called FICA UPDTUP2, consistent with control file processing from the bottom up.

Auxiliary control files can also be used with XEDIT. You can make your control file point to AUX files which contain the filetypes of the actual update files, or you can combine AUX files and update files in a single control file. XEDIT begins applying updates from the bottom up in the control file and references the AUX files indicated. Any updates to the source file produced during the editing session are inserted in the topmost update filetype specified in either the control file or in the last AUX file encountered using the 'bottom up' processing rule. More information about the XEDIT CTL option can be found in the *VM/SP System Product Editor Command and Macro Reference*.

Preferred Level Updating:

There may exist more than one version of an update, each applicable to different versions of the same module. For example, you may need one version of an update for an unmodified base source module, and another version of that update if that module has been modified by a program product. The AUX file that will be used to update a particular module must then be selected based on whether or not a program product modifies that module. The AUX files listing the updates applicable to modules modified by a program product are called "preferred AUX files" because they must be used if they exist rather than the mutually exclusive updates applicable to unmodified modules. Using this preferred AUX file concept, every module in a component can be assembled using the one CNTRL file applicable to a user's configuration.

A single AUX file entry in a CNTRL file can specify more than one filetype. The first filetype indicates a file that UPDATE uses only on one condition: the files that the second and subsequent filetypes indicate do not exist. If they do exist, this AUX file entry is ignored and no updating is done. The files that the second and subsequent filetypes indicate are preferred because, if they exist, UPDATE does not use the file that the first filetype indicates. Usually, the preferred files appear later in the CNTRL file in a format that causes them to be used for updating.

UPDATE scans each CNTRL file entry until a preferred filetype is found, until there are no more filetypes on the entry, or until a comment is found. (A character string that is less than four or more than eight characters is assumed to be a comment.)

The VMFASM EXEC Procedure

If you are an assembler language programmer and you are using the UPDATE command to update source programs you may want to use the VMFASM EXEC procedure. VMFASM is a VM/SP update procedure; it invokes the UPDATE command and then uses the ASSEMBLE command to assemble the updated source file.

If you are not an assembler language programmer, you may wish to create an EXEC similar to VMFASM that, instead of calling the assembler, calls one of the language compilers to compile an updated source file.

When you use VMFASM, you specify the source filename, the filename of the control file, and optionally, parameters for the assembler. (The control file for VMFASM must have a filetype of CNTRL). For example, if you use the file GENERAL CNTRL to update SAMPLE ASSEMBLE, you enter the command line:

```
vmfasm sample general
```

The VMFASM EXEC uses the MACS card and the update level identifiers in the control file. It reads the MACS card to determine which macro libraries (MACLIBs) should be searched by the assembler. Then VMFASM issues the GLOBAL MACLIB command specifying the MACLIBs you name on the MACS card.

The update level identifier is used by VMFASM to name the output text file produced by the assembly. If the update level identifier of the most recent update file (the last one located and applied) is anything other than TEXT, the update level identifier is prefixed with the characters TXT to form the filetype. For example, if the file GENERAL CNTRL contains the records:

```
TEXT MACS CMSLIB MYLIB OSMACRO
UP2 FIX2
UP1 FIX1
TEXT AUXLIST
```

and it is used to update the file SAMPLE ASSEMBLE, then:

- If the file SAMPLE UPDTFIX2 is found and the updates applied, VMFASM names the output text deck SAMPLE TXTUP2.
- If the file SAMPLE UPDTFIX1 is found and the updates applied but no SAMPLE UPDTFIX2 is found, the text deck is named SAMPLE TXTUP1.
- If the file SAMPLE AUXLIST is found but no SAMPLE UPDTFIX1 or SAMPLE UPDTFIX2 files are found, the text deck is named SAMPLE TEXT.
- If no files are found, the update level identifier on the MACS card is used and the text deck is named SAMPLE TEXT.

The new fn TEXT or fn TXTxxxxx resides on the A-disk. Because the UPDATE command works from the bottom of a control file toward the top, it is logical that the text filename be taken from the identifier of the last update applied.

The VMFASM EXEC does not produce an updated source file, but leaves the original source intact.

VMFASM produces two output files:

- a printed output listing that shows update activity
- the text file, which contains the update log as well as the actual object code.

If you use the CMS LOAD command to load a text file produced by VMFASM, records from the update log are flagged as invalid, but the LOAD operation is not impaired.

The STK Option:

If you are interested in writing your own EXEC procedure to invoke the UPDATE command, you may wish to use the STK option. The STK (stack) option is valid only with the CTL option, and is meaningful only when the UPDATE command is invoked within an EXEC procedure.

When the STK option is specified, UPDATE stacks the following data lines in the console stack:

```
first line:  * update level identifier
second line: * library list from MACS record
```

The update level identifier is the identifier of the most recent update that was found and applied.

For example, a CMS EXEC file that invokes the UPDATE command and then the ASSEMBLE command may contain the lines:

```
&TRACE ALL
UPDATE &1 ASSEMBLE * &2 CNTRL * (STK CTL
&READ VARS &STAR &TX
&READ VARS &STAR &LIB1 &LIB2 &LIB3 &LIB4 &LIB5 &LIB6 &LIB7 &LIB8
GLOBAL MACLIB &LIB1 &LIB2 &LIB3 &LIB4 &LIB5 &LIB6 &LIB7 &LIB8
&IF &TX NE TEXT FILEDEF TEXT DISK &1 TXT&TX A1
ASSEMBLE &1
ERASE $&1 ASSEMBLE
```

Below is a System Product interpreter EXEC that invokes the UPDATE command and then the ASSEMBLE command:

```
 /* Sample System Product interpreter EXEC to    */
 /* Update and Assemble a source program         */
 trace a
 parse arg filename cntrlfile .
'UPDATE' filename 'assemble *' cntrlfile 'cntrl * (STK CTL'
 parse pull star tx
 parse pull star lib1 lib2 lib3 lib4 lib5 lib6 lib7 lib8
'GLOBAL MACLIB' lib1 lib2 lib3 lib4 lib5 lib6 lib7 lib8
 if tx ¬= TEXT then 'FILEDEF TEXT DISK' filename 'XT'tx 'A1'
'ASSEMBLE $'filename
'ERASE  $'filename 'ASSEMBLE'
```

If the EXEC that you use is named UPASM EXEC, it is invoked with the line:

```
upasm fica fica (print noxref
```

and the file FICA CNTRL contains:

```
MAC MACS CMSLIB OSMACRO MYTEST
FIX1  UPDTFIX
LIST  AUXLIST
```

then the CMS EXEC executes the following commands:

```
UPDATE FICA ASSEMBLE * FICA CNTRL * (STK CTL
GLOBAL MACLIB CMSLIB OSMACRO MYTEST
FILEDEF TEXT DISK FICA TXTFIX1 A1
ASSEMBLE $FICA (PRINT NOXREF
ERASE $FICA ASSEMBLE
```

The System Product interpreter EXEC executes the following:

```
/* Update FICA ASSEMBLE using FICA CNTRL */
'UPDATE FICA ASSEMBLE * FICA CNTRL * (STK CTL'
'GLOBAL MACLIB CMSLIB OSMACRO MYTEST'
'FILEDEF TEXT DISK FICA TXTFIX1 A1'
'ASSEMBLE $FICA (PRINT NOXREF'
'ERASE $FICA ASSEMBLE'
```

The above examples assume that the update file FICA UPDTFIX was found and applied.

Chapter 9. Developing OS programs under CMS

CMS simulates many of the functions of the Operating System (OS), allowing you to compile, execute and debug OS programs interactively. For the most part, you do not need to be concerned with the CMS OS simulation routines; they are built into the CMS system. Before you can compile and execute OS programs in CMS, however, you must be acquainted with the following:

- OS macros that CMS can simulate
- Using OS data sets in CMS
- How to use the FILEDEF command
- Creating CMS files from OS data sets
- Using CMS and OS macro libraries
- Assembling program in CMS
- Executing programs

These topics are discussed below. Additional information for OS VSAM users is in Chapter 11, "Using Access Method Services and VSAM Under CMS and CMS/DOS" on page 11-1.

For a practice terminal session using the commands and techniques presented in this section, see Appendix F, "Sample Terminal Sessions."

> **Note:** The CMS system uses many OS terms, but there are a number of OS functions that CMS performs somewhat differently. Refer to Figure 9-1 on page 9-2 to help you become familiar with some of the equivalents (where they do exist) for OS terms and functions. It lists some commonly-used OS terms and discusses how CMS handles the functions they imply.

OS Term/Function	CMS Equivalent
Cataloged procedure	EXEC files can execute command sequences similar to cataloged procedures, and provide for conditional execution based on return codes from previous steps.
Data set	Data sets are called files in CMS. CMS can simulate certain OS data sets and can read real OS data sets only if they are sequential or partitioned. CMS can never write to real OS data sets. CMS reads and writes VSAM data sets.
Data Definition (DD) card	The FILEDEF command allows you to perform the functions of the DD statement to specify device types and output file dispositions.
Data Set Control Block (DSCB)	Information about a CMS disk file is contained in a file status table (FST).
EXEC card	To execute a program in CMS you specify only the name of the program if it is an EXEC, MODULE file, or CMS command. To execute TEXT files, use the LOAD and START commands.
Job Control Language (JCL)	CMS and user-written commands perform the functions of JCL.
Link-editing	The CMS LKED command creates LOADLIB libraries from CMS TEXT files and/or OS object modules. The CMS LOAD command loads TEXT files into virtual storage, and resolves external references; the GENMOD command creates absolute nonrelocatable modules.
Load module	Load modules are members of CMS LOADLIB libraries. LOADLIB members are loaded, relocated, and executed by the OSRUN and NUCXLOAD commands. Also, LOADLIB members are referenced by the LINK, LOAD, ATTACH and XCTL macros.
Object module	Language compiler output is placed in CMS files with a filetype of TEXT.
Partitioned data set	CMS MACLIBs, TXTLIBs, and LOADLIBs are the only CMS files that resemble partitioned data sets.
SETPCAT, JOBCAT	VSAM catalogs can be assigned for jobs or job steps in CMS by using the special ddnames IJSYSCT and IJSYSUC when identifying catalogs.
STEPLIB, JOBLIB	The GLOBAL command establishes macro, text, and LOADLIB libraries; you can indirectly provide job libraries by accessing and releasing CMS disks that contain the files and programs you need.
Utility program	Functions similar to those performed by the OS utility programs are provided by CMS commands.
Volume Table of Contents (VTOC)	The list of files on a CMS disk is contained in a file directory.

Figure 9-1. OS Terms and CMS Equivalents

Using OS Data Sets in CMS

You can have OS disks defined in your virtual machine configuration; they may be either entire disks or minidisks: their size and extent depends on their VM/SP directory entries. You can use partitioned and sequential data sets on OS disks in CMS. If you want, you can create CMS files from your OS data sets. If you have

data sets on OS disks, you can read them from programs you execute in CMS, but you cannot update them. The CMS commands that recognize OS data sets on OS disks are listed in Figure 9-2.

Command	Operation
ACCESS	Makes the OS disk containing the data set available to your CMS virtual machine.
ASSEMBLE	Assembles an OS source program under CMS.
DDR	Copies an entire OS disk to tape.
DLBL	Defines OS data sets for use with access method services and VSAM files for program input/output.
FILEDEF	Defines the OS data set for use under CMS by associating an OS ddname with an OS data set name. Once defined, the data set can be used by an OS program running under CMS and can be manipulated by the other commands that support OS functions.
GLOBAL	Makes macro libraries or LOADLIB libraries available to CMS. You can prepare an OS library for reference by the GLOBAL command by issuing a FILEDEF command for the data set and giving the data set the appropriate filetype of MACLIB or LOADLIB.
LKED	Creates CMS LOADLIB libraries from CMS TEXT files and or OS object modules.
LISTDS	Lists information describing OS data sets residing on OS disks.
MOVEFILE	Moves data records from one device to another device. Each device is specified by a ddname, which must have been defined via FILEDEF. You can use the MOVEFILE command to create CMS files from OS data sets.
NUCXLOAD	Loads, relocates, and establishes as a nucleus extension a load module either from a CMS LOADLIB , an OS module library or an OS formatted disk.
OSRUN	Loads, relocates, and executes a load module either from a CMS LOADLIB or from an OS module library on an OS formatted disk.
QUERY	Lists (1) the files that have been defined with the FILEDEF and DLBL commands (QUERY FILEDEF, QUERY DLBL), or (2) the status of OS disks attached to your virtual machine (QUERY DISK, QUERY SEARCH).
RELEASE	Releases an OS disk you have accessed (via ACCESS) from your CMS virtual machine.
STATE	Verifies the existence of an OS data set on a disk. Before STATE can verify the existence of the data set, you must have defined it (via FILEDEF).

Figure 9-2. CMS Commands that Recognize OS Data Sets on OS Disks

Access Methods Supported by CMS

OS access methods are supported, to varying extents, by CMS. Under CMS, you can execute programs that use the OS data management macros that are supplied for the access methods listed in Figure 9-3

Access Method	CMS Support for OS Simulated Data Sets on CMS Disks	CMS Support for Real OS Data Sets on OS Disks
BDAM	Yes	No
BPAM	Yes	Yes (read only)
BSAM	Yes	Yes (read only)
QSAM	Yes	Yes (read only)
VSAM	No	Yes

Figure 9-3. Access Methods Supported by CMS

`BPAM, BSAM, and QSAM:`
You can execute programs in CMS that read records from OS data sets using the BPAM, BSAM, or QSAM access methods. You cannot, however, write or update OS data sets that reside on OS disks.

`BDAM:`
CMS can neither read nor write OS data sets on OS disks using the BDAM access method.

`VSAM Files:`
CMS can read and write VSAM files on OS disks. For information on using VSAM under CMS, see Chapter 11, "Using Access Method Services and VSAM Under CMS and CMS/DOS" on page 11-1,

OS Simulated Data Sets

If you want to test programs in CMS that create or modify OS data sets, you can write "OS simulated data sets." These are CMS files that are maintained on CMS disks, but in OS format rather than in CMS format. Since they are CMS files, you can edit, rename, copy, or manipulate them just as you would any other CMS file. Since they are in OS-simulated format, files with variable-blocked records may contain block and record descriptor words so that the access methods can manipulate them properly.

The files that you create from OS programs do not necessarily have to be OS simulated data sets. You can create CMS files. The format of an output file depends on how you specify the filemode number when you issue the FILEDEF command to identify the file to CMS. If you specify the filemode number as 4, CMS creates a file that is in OS simulated data set format on a CMS disk. If you want to read an OS simulated dataset that is variable blocked or fixed blocked, rename the dataset with a filemode number of 4. CMS OS simulation routines are then able to read short blocks that are not filled with records.

CMS can read and write OS simulated data sets using the BDAM, BPAM, BSAM, and QSAM access methods.

When an input or output error occurs, do not depend on OS sense bytes. An error code is supplied by CMS in the ECB in place of the sense bytes. These error codes differ for various types of devices and their meaning can be found in the *VM/SP System Messages and Codes* under DMSxxx120S.

Note: Results may be unpredictable if two DCBs access the same data set at the same time.

Restrictions for Reading OS Data Sets

The following restrictions apply when you read OS data sets from OS disks under CMS:

- Read-password-protected data sets are not read.
- RACF password protection is ignored.
- BDAM and ISAM data sets are not read.
- Multivolume data sets are read as single-volume data sets. End-of-volume is treated as end-of-file and there is no end-of-volume switching.
- Keys in data sets with keys are ignored; only the data is read.
- User labels in user-labeled data sets are bypassed. See "Tape Labels in CMS" for details.
- Results may be unpredictable if two DCBs access the same data set at the same time.
- An Indexed VTOC on an OS disk is read the same as a standard OS VTOC since there is no special support in CMS for this.

Using the FILEDEF Command

Whenever you execute an OS program under CMS that has input and/or output files, or you need to read an OS data set onto a CMS disk, you must first identify the files to CMS with the FILEDEF command. The FILEDEF command in CMS performs the same functions as the data definition (DD) card in OS job control language (JCL): it describes the input and output files.

When you enter the FILEDEF command, you specify:

- The ddname
- The device type
- A file identification, if the device type is DISK
- Type of label on your tape file, if tape label processing is specified
- Options (if necessary)

Some guidelines for entering these specifications follow.

Specifying the ddname

If the FILEDEF command is issued for a program input or output file, then the ddname must be the same as the ddname or file name specified for the file in the source program. For example, you have an assembler language source program that contains the line:

```
INFILE   DCB   DDNAME=INPUTDD,MACRF=GL,DSORG=PS,RECFM=F,LRECL=80
```

For a particular execution of this program, you want to use as your input file a CMS file on your A-disk that is named MYINPUT FILE. You must then issue a FILEDEF for this file before executing the program:

```
filedef inputdd disk myinput file a1
```

If the input file you want to use is on an OS disk accessed as your C-disk, and it has a data set name of PAYROLL.RECORDS.AUGUST, then your FILEDEF command might be:

```
filedef inputdd c1 dsn payroll.records.august
```

Specifying the Device Type

For input files, the device type you enter on the FILEDEF command indicates the device from which you want records read. It can be DISK, TERMINAL, READER (for input from real cards or virtual cards), or TAPn (for tape). Using the above example, if your input file is to be read from your virtual card reader, the FILEDEF command might be as follows:

```
filedef inputdd reader
```

Or, if you were reading from a tape attached to your virtual machine at virtual address 181 (TAP1):

```
filedef inputdd tap1
```

For output files, the device you specify can be DISK, PRINTER, PUNCH, TAPn (tape), or TERMINAL.

If you do not want any real I/O performed during the execution of a program for a disk input or output file, you can specify the device type as DUMMY:

```
filedef inputdd dummy
```

Entering File Identifications

If you are using a CMS disk file for your input or output, you specify:

```
filedef ddname disk filename filetype filemode
```

> **Note:** If * is used for the filemode of an output file, unpredictable results may occur.

The filemode field is optional; if you do not specify it, your A-disk is assumed. If you want an output file to be constructed in OS simulated data set format, you must specify the filemode number as 4. For example, a program contains a DCB for an output file with a ddname of OUTPUTDD, and you are using it to create a CMS file named DAILY OUTPUT on your B-disk:

```
filedef outputdd disk daily output b4
```

If your input file is an OS data set on an OS disk, you can identify it in several ways:

- If the data set name has only two qualifiers, for example HEALTH.RECORDS, you can specify:

  ```
  filedef inputdd disk health records b1
  ```

- If it has more than two qualifiers, you can use the DSN keyword and enter:

  ```
  filedef inputdd b1 dsn health records august 1974
                  -- or --
  filedef inputdd b1 dsn health.records.august.1974
  ```

Or you can request a prompt for a complete data set name:

```
filedef inputdd b1 dsn ?
ENTER DATA SET NAME:
health.records.august.1974
```

Note: When you enter a data set name using the DSN keyword, either with or without a request for prompting, you should omit the device type specification of DISK, unless you want to assign a CMS file identifier, as in the example below.

- You can also relate an OS data set name to a CMS file identifier:

```
filedef inputdd disk ossim file c1 dsn monthly records
                    -- or --
filedef inputdd disk ossim file c1 dsn monthly.records
```

Then you can refer to the OS data set MONTHLY.RECORDS by using the CMS file identifier, OSSIM FILE:

```
state ossim file c
```

When you do not issue a FILEDEF command for a program input or output file, or if you enter only the ddname and device type on the FILEDEF command, such as:

```
filedef oscar disk
```

then CMS issues a default file definition, as follows:

```
FILEDEF ddname DISK FILE ddname A1
```

where ddname is the ddname you assigned in the DDNAME operand of the DCB macro in your program or on the FILEDEF command. For example, if you assign a ddname of OSCAR to an output file and do not issue a FILEDEF command before you execute the program, then the CMS file FILE OSCAR A1 is created when you execute the program. If the filetype of a CMS input file, FILE ddname A1, is the same as the assigned DDNAME, the file can be identified by a default file definition. Even though an input file can be defined explicitly or by default, if an attempt is made to read the file and the file is not found, unpredictable results my occur.

Specifying CMS Tape Label Processing

You can use the label operands on the FILEDEF command to indicate that CMS tape label processing is not desired (this is the default). If CMS tape label processing is desired you can use the label operands on the FILEDEF command to indicate the types of labels on your tape. See "Tape Labels in CMS" on page 6-11 for a description of CMS tape label processing.

Specifying Options

The FILEDEF command has many options; those mentioned below are a sampling only. For complete descriptions of all the options of the FILEDEF command, see the *VM/SP CMS Command and Macro Reference.*

BLOCK, LRECL, RECFM, DSORG

If you are using the FILEDEF command to relate a data control block (DCB) in a program to an input or output file, you may need to supply some of the file format information, such as the record length and block size, on the FILEDEF command line. For example, if you have coded a DCB macro for an output file as follows:

```
OUTFILE  DCB    DDNAME=OUT,MACRF=PM,DSORG=PS
```

then, when you are issuing a FILEDEF for this ddname, you must specify the format of the file. To create an output file on disk, blocked in OS simulated data set format, you could issue:

```
filedef out disk myoutput file a4 (recfm fb lrecl 80 block 1600
```

To punch the output file onto cards, you would issue:

```
filedef out punch (lrecl 80 recfm f
```

You must supply file format information on the FILEDEF command line whenever it is not supplied on the DCB macro, except for existing disk files. When the OPEN macro instruction is executed, the CMS simulation of the OS OPEN routine initializes the Data Control Block (DCB). The DCB fields are filled in with information from the DCB macro instruction, the information specified on the FILEDEF command, or if the data set already exists, the data set label. However, if more than one source specified information for a particular field, only one source is used.

The order in which the DCB fields are filled follows:

1. The DCB macro instruction in your program.
2. The fields you had specified on the FILEDEF command
3. The data set label if the data set already exists.

The DCB macro instruction takes precedence over the FILEDEF and the data set label. The FILEDEF takes precedence over the data set label.

Data set label information from an existing CMS file is used only when the OPEN is for input or update, otherwise, the OPEN routine erases the existing file. You can modify any DCB field either before the data set is opened or through a Data Control Block exit of EXIT LIST (EXLST) options. When the data set is closed, the Data Control Block is restored to its original condition. Fields that were merged in at OPEN time from the FILEDEF and the data set label are cleared.

PERM

Usually, when you execute one of the language processors, all existing file definitions are cleared. If the development of a program requires you to recompile and re-execute it frequently, you might want to use the PERM option when you issue file definitions for your input and output files. For example:

```
cp spool punch to *
filedef indd disk test file a1 (lrecl 80 perm
filedef outdd punch (lrecl 80 perm
```

In this example, since you spooled your virtual punch to your own virtual card reader, output files are placed in your virtual reader. You can either read or delete them.

All file definitions issued with the PERM option stay in effect until you log off, specifically clear those definitions, or redefine them:

```
filedef indd clear
filedef outdd tap1 (lrecl 80
```

In the above example, the definition for INDD is cleared; OUTDD is redefined as a tape file.

When you issue the command:

```
filedef * clear
```

all file definitions are cleared, except those you enter with the PERM option.

When a program abends, or when you issue the HX Immediate command, all file definitions are cleared, including those entered with the PERM option.

DISP MOD

When you issue a FILEDEF command for an output file and assign it a CMS file identifier that is identical to that of an existing CMS file, then when anything is written to that ddname the existing file is replaced by the new output file. If you want, instead, to have new records added to the bottom of the existing file, you can use the DISP MOD option:

```
filedef outdd disk new update a1 (disp mod
```

The file must be on a disk accessed as read/write. Note that an extension of a disk is read/only. When adding new records using the DISP MOD option, erase the end-of-file (EOF) mark at the end of the existing file for fixed-block (FB) OS simulated files (filemode of A4).

MEMBER

If the file you want to read is a member of an OS partitioned data set (or a CMS MACLIB or TXTLIB), you can use the MEMBER option to specify the membername; for example:

```
filedef test c dsn sys1.maclib (member test
```

defines the member TEST from the OS macro library SYS1.MACLIB.

AUXPROC

This option allows an auxiliary processing routine to receive control during I/O operations. It is valid only when FILEDEF is executed by an internal program call and cannot be entered on a terminal command. For details on how to use this option of the FILEDEF command, see the *VM/SP System Programmer's Guide.*

Creating CMS Files From OS Data Sets

If you have data sets on OS disks, or on tapes or cards, you can copy them into CMS files so that you can edit, modify, or manipulate them with CMS commands. The CMS MOVEFILE command copies OS (or CMS) files from one device to another. You can move data sets from any valid input device to any valid output device.

Before using the MOVEFILE command, you must define the input and output data sets or files and assign them ddnames using the FILEDEF command. If you use the ddnames INMOVE and OUTMOVE, then you do not need to specify the ddnames when you issue the MOVEFILE command. For example, the following sequence of commands copies a CMS disk file into your virtual card punch:

```
filedef inmove disk diskin file a1
filedef outmove punch
movefile
```

The result of these commands is effectively the same as if you had issued the command:

```
punch diskin file (noheader
```

The example does, however, illustrate the basic relationship between the FILEDEF and MOVEFILE commands. In addition to the MOVEFILE command, if the OS input data set is on tape or cards, you can use the TAPPDS or READCARD command to create CMS files. These are also discussed below.

> **Note:** The MOVEFILE command does not support data containing spanned records.

Copying Sequential Data Sets from Disk

The MOVEFILE command copies a sequential OS disk data set from a read-only OS disk into an integral CMS file on a CMS read/write disk. You use FILEDEF commands to identify the input file disk mode and data set name:

```
filedef inmove c1 dsn sales.manual
```

the CMS output file's disk location and fileid:

```
filedef outmove disk sales manual a1
```

and then you issue the MOVEFILE command:

```
movefile
```

Copying Partitioned Data Sets From Disk

The MOVEFILE command can copy partitioned data sets (PDS) into CMS disk files, and create separate CMS files for each member of the data set. You can have the entire data set copied, or you can copy only a selected member. For example, if you have a partitioned data set named ASSEMBLE.SOURCE whose members are individual assembler language source files, your input file definition might be:

```
filedef inmove c1 dsn assemble source
           -- or --
filedef inmove c1 dsn assemble.source
```

To create individual CMS ASSEMBLE files, you would issue the output file definition as:

```
filedef outmove disk qprint assemble a1
```

Then use the PDS option of the MOVEFILE command:

```
movefile (pds
```

When the CMS files are created, the filetype on the output file definition is used for the filetype and the member names are used instead of the CMS filename you specified.

If you want to copy only a single member, you can use the MEMBER option of the FILEDEF command:

```
filedef inmove disk assemble source c (member qprint
```

and omit the PDS option on the MOVEFILE command:

```
movefile
```

Figure 9-4 on page 9-12 summarizes the various ways that you can create CMS files from OS data sets.

Using CMS Libraries

CMS provides three types of libraries to aid in OS program development:

- Macro libraries contain macro definitions and/or copy files
- Text, or program libraries contain relocatable object programs (compiler output)
- LOADLIB libraries contain link edit files (load modules)

These CMS libraries are like OS partitioned data sets; each has a directory and members. Since they are not like other CMS files, you create, update, and use them differently than you do other CMS files. Although these library files are similar in function to OS partitioned data sets, OS macros should not be used to update them. Macro libraries are discussed below; text libraries are discussed under "TEXT Libraries (TXTLIBs)," and LOADLIB libraries are discussed under "Executing Members of OS Modules Libraries or CMS LOADLIBS."

A CMS macro library has a filetype of MACLIB. You can create a MACLIB from files with filetypes of MACRO or COPY. A MACRO file may contain macro definitions; COPY files contain predefined source statements.

```
Input File:  An OS sequential data set named: COMPUTE.TEST.RECORDS
-------------------------------------------------------------------------------
 Source | CMS Command Examples                                  | CMS Output File
--------------------------------------------------------------- ---------------
Disk:   | filedef indd cl dsn compute test records              | COMPUTE RECORDS A1
 OS R/O | filedef outdd disk compute records a1                 |
 C-disk | movefile indd outdd                                   |
--------------------------------------------------------------------------------
Tape:   | filedef inmove tap1 (lrecl 80                         | TEST RECORDS A1
 181    | filedef outmove disk test records a1                  |
        | movefile                                              |
        |-------------------------------------------------------------------
        | tappds newtest compute (nopds                         | NEWTEST COMPUTE A1
--------------------------------------------------------------------------------
Cards   | filedef cardin reader                                 | COMPUTE CARDS A1
        | filedef diskout disk compute cards a1                 |
        | movefile cardin diskout                               |
        |-------------------------------------------------------------------
        | readcard compute test                                 | COMPUTE TEST A1
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Input file: OS partitioned data set named: TEST.CASES
                  Members named: SIMPLE, COMPLEX, MIXED
--------------------------------------------------------------------------------
 Source | CMS Command Examples                                  | CMS Output File(s)
--------------------------------------------------------------------------------
Disk:   | filedef infile disk test cases c1                     | SIMPLE TESTCASE A1
 OS R/O | filedef outfile disk new testcase a1                  | COMPLEX TESTCASE A1
 C-disk | movefile infile outfile (pds                          | MIXED TESTCASE
        |-------------------------------------------------------------------
        | filedef in cl dsn test cases (member simple           | FILE RUN A1
        | filedef run disk                                      |
        | movefile in run                                       |
--------------------------------------------------------------------------------
Tape:   | tappds * testrun  (tap2                               | SIMPLE TESTRUN A1
 182    |                                                       | COMPLEX TESTRUN A1
        |                                                       | MIXED TESTRUN A1
```

Figure 9-4. Creating CMS Files from OS Data Sets

When you want to assemble or compile a source program that uses macro or copy definitions, you must ensure that the library containing the code is identified before you invoke the compiler. Otherwise, the library is not searched. You identify libraries to be searched using the GLOBAL command. For example, if you have two MACLIBs that contain your private macros and copy files whose names are TESTMAC MACLIB and TESTCOPY MACLIB, you would issue the command:

```
global maclib testmac testcopy
```

The libraries you specify on a GLOBAL command line are searched in the order you specify them. A GLOBAL command remains in effect for the remainder of your terminal session, until you issue another GLOBAL MACLIB command or IPL CMS again. To find out what macro libraries are currently available for searching, issue the command:

```
query maclib
```

You can reset the libraries or the search order by reissuing the GLOBAL command.

The MACLIB Command

The MACLIB command performs a variety of functions. You use it to:

- Create the MACLIB (GEN function)
- Add, delete, or replace members (ADD, DEL, and REP functions)
- Compress the MACLIB (COMP function)
- List the contents of the MACLIB (MAP function)

Descriptions of these MACLIB command functions follow.

GEN Function

The GEN (generate) function creates a CMS macro library from input files specified on the command line. The input files must have filetypes of either MACRO or COPY. For example:

```
maclib gen osmac access time put regequ
```

creates a macro library with the file identifier OSMAC MACLIB A1 from macros existing in the files with the file identifiers:

ACCESS {MACRO / COPY}, TIME {MACRO / COPY}, PUT {MACRO / COPY}, and REGEQU {MACRO / COPY}

If a file named OSMAC MACLIB A1 already exists, it is erased.

Assume that the files ACCESS MACRO, TIME COPY, PUT MACRO, and REGEQU COPY exist and contain macros in the following form:

```
ACCESS MACRO     TIME COPY          PUT MACRO      REGEQU COPY

   GET           *COPY TTIMER         PUT            XREG
                  TTIMER
   PUT           *COPY STIMER                        YREG
                  STIMER
```

The resulting file, OSMAC MACLIB A1, contains the members:

```
GET        STIMER
PUT        PUT
TTIMER     REGEQU
```

The PUT macro, which appears twice in the input to the command, also appears twice in the output. The MACLIB command does not check for duplicate macro names. If, at a later time, the PUT macro is requested from OSMAC MACLIB, the first PUT macro encountered in the directory is used.

When COPY files are added to MACLIBs, the name of the library member is taken from the name of the COPY file, or from the *COPY statement, as in the file TIME COPY, above.

> **Note:** Although the file REGEQU COPY contained two macros, they were both included in the MACLIB with the name REGEQU. When the input file is a MACRO file, the member name(s) are taken from macro prototype statements in the MACRO file.

ADD Function

The ADD function appends new members to an existing macro library. For example, assume that OSMAC MACLIB A1 exists as created in the example in the explanation of the GEN function and the file DCB COPY exists as follows:

```
*COPY DCB
 DCB macro definition
*COPY DCBD
 DCBD macro definition
```

If you issue the command:

```
maclib add osmac dcb
```

the resulting OSMAC MACLIB A1 contains the members:

```
GET       PUT
PUT       REGEQU
TTIMER    DCB
STIMER    DCBD
```

REP Function

The REP (replace) function deletes the directory entry for the macro definition in the files specified. It then appends new macro definitions to the macro library and creates new directory entries. For example, assume that a macro library MYMAC MACLIB contains the members A, B, and C, and that the following command is entered:

```
maclib rep mymac a c
```

The files represented by file identifiers A MACRO and C MACRO each have one macro definition. After execution of the command, MYMAC MACLIB contains members with the same names as before, but the contents of A and C are different.

DEL Function

The DEL (delete) function removes the specified macro name from the macro library directory and compresses the directory so there are no unused entries. The macro definition still occupies space in the library, but since no directory entry exists it cannot be accessed or retrieved. If you attempt to delete a macro for which two macro definitions exist in the macro library, only the first one encountered is deleted. For example:

```
maclib del osmac get put ttimer dcb
```

deletes macro names GET, PUT, TTIMER, and DCB from the directory of the macro library named OSMAC MACLIB. Assume that OSMAC exists as in the ADD function example. After the above command, OSMAC MACLIB contains the following members:

```
STIMER
PUT
REGEQU
DCBD
```

COMP Function

Execution of a MACLIB command with the DEL or REP functions can leave unused space within a macro library. The COMP (compress) function removes any macros that do not have directory entries. This function uses a temporary file named MACLIB CMSUT1. For example, the command:

```
maclib comp mymac
```

compresses the library MYMAC MACLIB.

MAP Function

The MAP function creates a list containing the name of each macro in the directory, the size of the macro, and its position within the macro library. If you want to display a list of the members of a MACLIB at the terminal, enter the command:

```
maclib map mylib (term
```

The default option, DISK, creates a file on your A-disk, which has a filetype of MAP and a filename corresponding to the filename of the MACLIB. If you specify the PRINT option, the list is spooled to your virtual printer.

Note: The TERM and PRINT options will erase the old MAP file.

Manipulating MACLIB Members

The following CMS commands have MEMBER options, which allow you to reference individual members of a MACLIB:

- PRINT (to print a member)
- PUNCH (to punch a member)
- TYPE (to display a member)
- FILEDEF (to establish a file definition for a member)

You can use the CMS editor to create MACRO and COPY files and then use the MACLIB command to place the files in a library. Once they are in a library, you can erase the original files.

To extract a member from a macro library, you can use either the PUNCH or the MOVEFILE command. If you use the PUNCH command you can spool your virtual card punch to your own virtual reader:

```
cp spool punch to *
```

Then punch the member:

```
punch testmac maclib (member get noheader
```

and read it back onto disk:

```
readcard get macro
```

In the above example, the member was punched with the NOHEADER option of the PUNCH command, so that a name could be assigned on the READCARD command line. If a header card had been created for the file, it would have indicated the filename and filetype as GET MEMBER.

If you use the MOVEFILE command, you must issue a file definition for the input member name and the output macro or copy name before entering the MOVEFILE command:

```
filedef inmove disk testcopy maclib (member enter
filedef outmove disk enter copy a
movefile
```

This example copies the member ENTER from the macro library TESTCOPY MACLIB into a CMS file named ENTER COPY.

When you use the PUNCH or MOVEFILE commands to extract members from CMS MACLIBs, each member is followed by a // record, which is a MACLIB delimiter. You can edit the file and use the DELETE subcommand to delete the // record.

If you wish to move the complete MACLIB to another file, use the COPYFILE command.

System MACLIBs

The macro libraries that are on the system disk contain CMS and OS assembler language macros that you may want to use in your programs:

- CMSLIB MACLIB contains the CMS macros from VM/370.
- DMSSP MACLIB contains the macros that are new or changed in VM/SP.

  Note: When assembling programs that use CMS macros, both of these libraries should be identified via the GLOBAL command. DMSSP should precede CMSLIB in the search order.

- OSMACRO MACLIB contains the OS macros that CMS supports or simulates or those that require no CMS support.
- OSMACRO1 MACLIB contains the macros CMS does not support or simulate. (You can assemble programs in CMS that contain these macros, but you must execute them in an OS virtual machine.)
- OSVSAM MACLIB contains the subset of supported OS/VSAM macros.
- TSOMAC MACLIB contains TSO macros.
- DOSMACRO MACLIB contains macros used internally in CMS/DOS.

  Note: The DOSMACRO MACLIB contains macros used internally by CMS/DOS system routines. These macros should not be used in user written programs.

To obtain a list of the macros in any of these libraries, use the MAP function of the MACLIB command.

Using OS Macro Libraries

If you want to assemble source programs that contain macro statements that are defined in macro libraries on your OS disks, you can use the FILEDEF command to identify them to CMS so that you can name them when you issue the GLOBAL command. For example, the commands:

```
filedef cmslib disk temp maclib c dsn test asm macros
global maclib temp
```

allow you to access the macro library TEST.ASM.MACROS on the OS disk accessed as your C-disk.

When you issue a FILEDEF command for an assembler language macro library you must use a ddname of CMSLIB and you must provide a CMS file identifier for the OS data set. In the example above, the OS macro library TEST.ASM.MACROS is given the CMS file identifier TEMP MACLIB.

If you want to use more than one OS macro library you must issue a FILEDEF command for each library using the ddname CMSLIB and specifying the CONCAT option. For example:

```
filedef cmslib disk asp1 maclib * dsn asp1.macros.r1 (concat recfm fb block 3360 lrecl 80
filedef cmslib disk asp2 maclib * dsn asp2.macros.r1 (concat
filedef cmslib disk sys1 maclib * (concat
global maclib asp1 asp2 sys1 osmacro cmslib
```

The GLOBAL command establishes the search order of the libraries as: ASP1.MACROS.R1, ASP2.MACROS.R1, SYS1.MACLIB, OSMACRO MACLIB, and CMSLIB MACLIB.

> **Note:** The third library specified is entered in an abbreviated form. You can use this form when the data set name of the macro library has only two qualifiers and the second qualifier is MACLIB; thus, the equivalency is established between SYS1.MACLIB and the CMS file identifier SYS1 MACLIB.

The file format information describes the macro libraries to CMS; when you are concatenating OS macro libraries, they must all be in the same format, since the options entered on the first FILEDEF command are applied to all the libraries.

Also, if you want to use the FILEDEF option CONCAT, the first FILEDEF command for concatenated macro libraries should describe the first library in the GLOBAL command. When a concatenated macro library is closed after use, the CMS filename in the FCB is restored to the first name in the global list. If this is not the one specified on the original FILEDEF, subsequent use may cause errors. Reissue the FILEDEF command.

If you are using only one OS macro library in addition to CMS MACLIBs you can enter:

```
filedef cmslib disk lib1 maclib * dsn sys1.maclib
global maclib lib1 cmslib
```

To identify libraries for use with the language processors, you must use the ddname SYSLIB.

Using the CONCAT option for the first FILEDEF (with PERM option) for concatenated libraries may cause errors if the FILEDEF is not cleared before subsequent use of FILEDEF.

Using OS Macro Simulation Under CMS

CMS provides routines that simulate the OS functions required to support OS language processors and user-written programs. CMS functionally simulates the OS macros in such a way that equivalent results are presented to programs executing under CMS.

Figure 9-5 on page 9-19 lists the OS macros and their functions that CMS partially or completely simulates. The macros that are listed as "effective no-op" and "no-op" are macros that are not supported in CMS; you can assemble programs

that contain these macros. However, when you execute them in CMS, the macro functions are not performed. To execute these programs, you must run them in an OS virtual machine.

For a more detailed description of how CMS simulates the functions of these macros, and to see whether any particular function of a macro is not supported, see the *VM/SP System Programmer's Guide.*

OS Data Management Simulation

A CMS file produced by an OS program running under CMS and written on a CMS disk, has a different format than that of an OS data set produced by the same OS program running under OS and written on an OS disk. The data is the same, but, the format is different. CMS can read, write, or update any OS data that resides on a CMS disk.

Macro	SVC No.	Function
ABEND	13	Terminate processing
ATTACH	42	Effective LINK
BLDL	18	Build a directory list for a PDS
BSP	69	Back up a record on a tape or disk
CHAP	44	Effective no-op
CHECK	-	Verify READ/WRITE macro completion
CHKPT	63	Effective no-op
CLOSE	20	Deactivate a data file
DCB	-	Construct a data control block
DCBD	-	Generate a DSECT for a data control block
DELETE	09	Delete a loaded phase
DEQ	48	Effective no-op
DETACH	62	Effective no-op
DEVTYPE	24	Obtain device-type characteristics
ENQ	56	Effective no-op
EXIT/RETURN	03	Return from called phase
EXTRACT	40	Effective no-op
FEOV	31	Set forced EOV error code
FIND	18	Locate a member of a partitioned data set
FREEDBUF	57	Release a free storage buffer
FREEMAIN	05	Release user-acquired storage
FREEMAIN	10	Manipulate user free storage
FREEPOOL	-	Simulate as SVC 10
GET	-	Read system-blocked data (QSAM)
GETMAIN	04	Conditionally acquire user storage
GETMAIN	10	Manipulate user free storage
GETPOOL	-	Simulate as SVC 10
IDENTIFY	41	Add entry to loader table
LINK	06	Link control to another phase
LOAD	08	Read a phase into storage
NOTE	-	Manage data set positioning
OPEN	19	Activate a data file
OPENJ	22	Activate a data file
PGRLSE	112	Release storage contents
POINT	-	Manage data set positioning
POST	02	Post the I/O completion
PUT	-	Write system-blocked data (QSAM)

Figure 9-5 (Part 1 of 2). OS Macros Simulated by CMS

Macro	SVC No.	Function
RDJFCB	64	Obtain information from FILEDEF command
READ	-	Access system-record data
RESTORE	17	Effective no-op
RETURN	-	Return from a subroutine
SAVE	-	Save program registers
SNAP	51	Dump specified areas of storage
SPIE	14	Allow processing program to handle program interrupts
STAE	60	Allow processing program to decipher abend conditions
STAX	96	Create an attention exit block
STIMER	47	Set timer
STOW	21	Manipulate partitioned directories
SYNADAF	-	Provide SYNAD analysis function
SYNADRLS	-	Release SYNADAF message and save areas
TCLEARQ	94	Clear terminal input queue
TCLOSE	23	Temporarily deactivate a data file
TGET/TPUT	93	Read or write a terminal line
TIME	11	Get the time of day
TRKBAL	25	no-op
TTIMER	46	Access or cancel timer
WAIT	01	Wait for an I/O completion
WRITE	-	Write system-record data
WTO/WTOR	35	Communicate with the terminal
XCTL	07	Delete, then link control to another load phase
XDAP	00	Read or write direct access volumes

Figure 9-5 (Part 2 of 2). OS Macros Simulated by CMS

Assembling Programs in CMS

To assemble assembler language source programs into object module format, you can use the ASSEMBLE command, and specify assembler options on the command line; for example:

```
assemble myfile (print
```

assembles a source program named MYFILE ASSEMBLE and directs the output listing to the printer. All of the ASSEMBLE command options are listed in the *VM/SP CMS Command and Macro Reference*.

When you invoke the ASSEMBLE command specifying a file with the filetype of ASSEMBLE, CMS searches all of your accessed disks, using the standard search order, until it locates the specified file. When the assembler creates its output listing and text deck, it creates files with filetypes of LISTING and TEXT, and writes them onto disk according to the following priorities:

1. If the source file is on a read/write disk, the TEXT and LISTING files are written onto that disk.

2. If the source file is on a read-only extension of a read/write disk, the TEXT and LISTING files are written onto the parent disk.

3. If the source file is on any other read-only disk, the TEXT and LISTING files are written onto the A-disk.

4. If none of the above choices are available, the command is terminated.

In all of the above cases, the TEXT and LISTING files have a filename that is the same as the input ASSEMBLE file.

The input and output files used by the assembler are assigned by FILEDEF commands that CMS issues internally when the assembler is invoked. If you issue a FILEDEF command using one of the assembler ddnames before you issue the ASSEMBLE command, you can override the default file definitions.

The ddname for the source input file (SYSIN) is ASSEMBLE. If you enter:

```
filedef assemble reader
assemble sample
```

then the assembler reads your input file from your card reader, and assigns the filename SAMPLE to the output TEXT and LISTING files.

You could assemble a source file directly from an OS disk by entering:

```
filedef assemble disk myfile assemble b4 dsn os source file
assemble myfile
```

In this example, the CMS file identifier MYFILE ASSEMBLE is assigned to the data set OS.SOURCE.FILE and then assembled.

LISTING and TEXT are the ddnames assigned to the SYSPRINT and SYSLIN output of the assembler. You might assign file definitions to override these defaults as follows:

```
filedef listing disk assemble listfile a
filedef text disk assemble textfile a
assemble myfile
```

In this example, output from the assembly of the file, myfile ASSEMBLE, is written to the files, ASSEMBLE LISTFILE and ASSEMBLE TEXTFILE.

The ddnames PUNCH and CMSLIB are used for SYSPUNCH and SYSLIB data sets. PUNCH output is produced when you use the DECK option of the ASSEMBLE command. The default file definition for CMSLIB is the macro library CMSLIB MACLIB, but you must still issue the GLOBAL command if you want to use it.

Executing Programs

After you have assembled or compiled a source program you can execute the TEXT files that were produced by the assembly or compilation. You may not,

however, be able to execute all your OS programs directly in CMS. There are a number of execution-time restrictions placed on your virtual machine by VM/SP. You cannot execute a program that uses:

- Multitasking
- More than one partition
- Teleprocessing
- ISAM macros to read or write files
- Sets the EC mode bit in the PSW

The above is only a partial list, representing those restrictions with which you might be concerned. For a complete list of restrictions, see the *VM/SP Planning Guide and Reference*.

Executing TEXT Files

TEXT files, in CMS, are relocatable, and can be executed simply by loading them into virtual storage with the LOAD command and using the START command to begin execution. For example, if you have assembled a source program named CREATE, you have a file named CREATE TEXT. You can issue the command:

```
load create
```

which loads the relocatable object file into storage, and then, to execute it, you can issue the START command:

```
start
```

In the case of a simple program, as in the above example, you can load and begin execution with a single command line, using the START option of the LOAD command:

```
load create (start
```

When you issue the START command or LOAD command with the START option, control is passed to the first entry point in your program. If you have more than one entry point and you want to begin execution at an entry point other than the first, you can specify the alternate entry point or CSECT name on the START command:

```
start create2
```

When you issue the LOAD command specifying the filename of a TEXT file, CMS searches all of your accessed disks for the specified file.

If your program expects a parameter list to be passed (via register 1), you can specify the arguments on the START command line. If you enter arguments, then you must specify the entry point:

```
start * name1
```

When you specify the entry point as an asterisk (*) it indicates that you want to use the default entry point.

Defining Input and Output Files

You can issue the FILEDEF command to define input and output files any time before you begin program execution. You can issue all your file definitions before

loading any TEXT files, or issue them during the loading process. You can find out what file definitions are currently in effect by issuing the FILEDEF command with no operands:

```
filedef
```

You can also use the FILEDEF operand of the QUERY command.

TEXT Libraries (TXTLIBS)

You may want to keep your TEXT files in text libraries, that have a filetype of TXTLIB. Like MACLIBs, TXTLIBs have a directory and members. TXTLIBs are created and modified by the TXTLIB command, which has functions similar to the MACLIB command:

- The GEN function creates the TXTLIB.
- The ADD function adds members to the TXTLIB.
- The DEL function deletes members and compresses the TXTLIB.
- The MAP function lists members.

There is no REP function; you must use a DEL followed by an ADD to replace an existing member. The CMS commands that recognize MACLIBs as special filetypes also recognize TXTLIBs, and allow you to display, print, or punch TXTLIB members.

The TXTLIB command reads the object files as it writes them into the library, and creates a directory entry for each entry point or CSECT name. If you have a TEXT file named MYPROG, which has a single routine named BEGIN, and create the TXTLIB named TESTLIB as follows:

```
txtlib gen testlib myprog
```

TESTLIB contains no entry for the name MYPROG; you must specify the membername BEGIN to reference this TXTLIB member.

When you want to load members of TXTLIBs into storage to execute them (just as you execute TEXT files), you must issue the GLOBAL command to identify the TXTLIB:

```
global txtlib testlib
load begin (start
```

When you specify more than one TXTLIB on the GLOBAL command line, the order of search is established for the TXTLIBs. However, if the AUTO option is in effect (it is the default), CMS searches for TEXT files before searching active TXTLIBs.

When the TXTLIB command processes a TEXT file, it writes an LDT (loader terminate) card at the end of the TEXT file, so that when a load request is issued for a TXTLIB member, loading terminates at the end of the member. If you add OS linkage editor control statements to the TEXT file (using the CMS editor) before you issue the TXTLIB command to add the file to a TXTLIB, the control statements are processed as follows:

NAME Statement

A NAME statement causes the TXTLIB command to create the directory entry for the member using the specified name. Thereafter, when you want to load that member into storage or delete it from the TXTLIB you must refer to it by the name specified on the NAME statement.

The loader does not use name cards to resolve entry points. It is important that the name on the name card be the same as the name on the CSECT or entry card. This will ensure that the loader will find the correct text deck and loader tables (any external references) will be resolved with the entry point. If the names differ, the loader will load the text deck based on the name card (or file name). However, the loader tables will be set up according to entry or CSECT cards encountered during the load. Any external reference using the name from the name card will be resolved as zeros. See the section "Resolving External References" for more detailed information.

ENTRY Statement

If you use an ENTRY statement, the entry point you specify is validated and checked for a duplicate. If the entry point name is valid and there are no duplicates in the TEXT file, the entry name is written in the LDT card. Otherwise, an error message is issued. When this member is loaded, execution begins at the entry point specified. (See "Determining Program Entry Points.")

ALIAS Name

An entry is created in the directory for the ALIAS name you specify. A maximum of 16 alias names can be used in a single text deck. You may load the single member and execute it by referring to the alias name, but you cannot use the alias name as the object of V-type address constant (VCON), because the address of the member cannot be resolved.

SETSSI Card

Information you specify on the SETSSI card is written in bytes 26 through 33 of the LDT card.

All other OS linkage editor control statements are ignored by the TXTLIB command and written into the TXTLIB member. When you attempt to load the member, the CMS loader flags these cards as invalid.

Resolving External References

The CMS loader loads files into storage as a result of a LOAD or INCLUDE command. When a file is loaded, the loader checks for unresolved references; if there are any, the loader searches your disks for TEXT files with filenames that match the external entry name. When it finds a match, it loads the TEXT file into storage. If a TEXT file is not found, the loader searches any available TXTLIBs for members that match; if a match is found, it loads the member.

If there are still unresolved references, for example, if you load a program that calls routines PRINT and ANALYZE but the loader cannot locate them, you receive the message:

```
THE FOLLOWING NAMES ARE UNDEFINED:
 PRINT
 ANALYZE
```

You can issue the INCLUDE command to load additional TEXT files or TXTLIB members into storage so the loader can resolve any remaining references. For example, if you did not identify the TXTLIB that contains the routines you want to call, you may enter the GLOBAL command followed by the INCLUDE command:

```
global txtlib newlib
include print analyze (start
```

A failure to resolve external references might occur if you have TEXT files with filenames that are different from either the CSECT names or the entry names; You must explicitly issue LOAD and INCLUDE commands for these files.

At execution time, if there are still any unresolved references, their addresses are all set to 0 by the loader, so any attempt to address them in a program may result in a program check.

The LOAD and INCLUDE Commands

The INCLUDE command has the same format and option list (with one exception) as the LOAD command. The main difference is that when you issue the INCLUDE command the loader tables are not reset; if you issue two LOAD commands in succession, the second LOAD command cancels the effect of the first, and the pointers to the files loaded are lost.

Conversely, the INCLUDE command, which you must issue when you want to load additional files into storage, should not be used unless you have just issued a LOAD command. You may specify as many INCLUDE commands as necessary following a LOAD command to load files into storage.

Controlling the CMS Loader

The LOAD and INCLUDE commands allow you to specify a number of options. You can:

- Change the entry point to which control is to be passed when execution begins (RESET option).
- Specify the location in virtual storage at which you want the files to be loaded (ORIGIN option).
- Control how CMS resolves references and handles duplicate CSECT names (AUTO, LIB, and DUP options).
- Clear storage to binary zeros before loading files (CLEAR option). Otherwise CMS does not clear user storage.

When the LOAD and INCLUDE commands execute, they produce a load map, indicating the entry points loaded and their virtual storage locations. You may find this load map useful in debugging your programs. If you do not specify the NOMAP option, the load map is written onto your A-disk, in a file named LOAD MAP A5. Each time you issue the LOAD command, the old file LOAD MAP is erased and the new load map replaces it. If you do not want to produce a load map, specify the NOMAP option.

You can find details about these, and other options under the discussion of the LOAD command in *VM/SP CMS Command and Macro Reference*.

Loader Control Statements

In addition to the options provided with the LOAD and INCLUDE commands that assist you in controlling the execution of TEXT files, you can also use loader control statements. These can be inserted in TEXT files, using the CMS editor. The loader control statements allow you to:

- Set the location counter to specify the address at which the next TEXT file is to be loaded (SLC statement).
- Modify instructions and constants in a TEXT file, and change the length of the TEXT file to accommodate modifications (Replace and Include Control Section statements).
- Change the entry point (ENTRY statement).
- Nullify an external reference so that it does not receive control when it is called, and you do not receive an error message when it is encountered (LIBRARY statement).

These statements are also described under the LOAD command in *VM/SP CMS Command and Macro Reference*.

Determining Program Entry Points

When you load a single TEXT file or a TXTLIB member into storage for execution, the default entry point is the first CSECT name in the object file loaded. You can specify a different entry point at which to start execution either on the LOAD (or INCLUDE) command line with the RESET option:

```
load myprog (reset beta
```

where BETA is the alternate entry point of your program, or you can specify the entry point on the START command line:

```
start beta
```

When you load multiple TEXT files (either explicitly or implicitly, by allowing the loader to resolve external references), you also have the option of specifying the entry point on the LOAD, INCLUDE, or START command lines.

If you do not specifically name an entry point, the loader determines the entry point for you, according to the following hierarchy:

1. An entry point specified on the START command
2. The last entry specified with the RESET option on a LOAD or INCLUDE command
3. The name on the last ENTRY statement that was read
4. The name on the last LDT statement that contained an entry name that was read
5. The name on the first assembler- or compiler-produced END statement that was read

6. The first byte of the first control section loaded

For example, if you load a series of TEXT files that contain no control statements, and do not specify an entry point on the LOAD, INCLUDE, or START commands, execution begins with the first file that you loaded. If you want to control the execution of program subroutines, you should be aware of this hierarchy when you load programs or when you place them in TXTLIBs.

An area of particular concern is when you issue a dynamic load (with the OS LINK, LOAD, or XCTL macros) from a program, and you call members of CMS TXTLIBs. The CMS loader determines the entry point of the called program and returns the entry point to your program. If a TXTLIB member that you load has a VCON to another TXTLIB member, the LDT card from the second member may be the last LDT card read by the loader. If this LDT card specifies the name of the second member, then CMS may return that entry point address to your program, rather than the address of the first member.

Creating Program Modules

When your programs are debugged and tested, you can use the LOAD and INCLUDE commands, in conjunction with the GENMOD command, to create program modules. A module is a nonrelocatable file whose external references have been resolved. In CMS, these files must have a filetype of MODULE.

To create a program module, load the TEXT files or TXTLIB members into storage and issue the GENMOD command:

```
load create analyze print
genmod process
```

In this example, PROCESS is the filename you are assigning the module; it will have a filetype of MODULE. You could use any name; if you use the name of an existing MODULE file, the old one is replaced.

To execute the program composed of the source files CREATE, ANALYZE, and PRINT, enter:

```
process
```

If PROCESS requires input and/or output files, you will have to define these files before PROCESS can execute properly; if PROCESS expects arguments passed to it, you can enter them following the MODULE name; for example:

```
process test1
```

For more information on creating program modules, see Chapter 8, "Programming for The CMS Environment."

Using EXEC Procedures

During your program development and testing cycle, you may want to create EXEC procedures to contain sequences of CMS commands that you execute frequently. For example, if you need a number of MACLIBs, TXTLIBs, and file definitions to execute a particular program, you might have an EXEC procedure as follows:

```
&CONTROL ERROR TIME
&ERROR &EXIT &RETCODE
GLOBAL MACLIB TESTLIB OSMACRO OSMACRO1
ASSEMBLE TESTA
PRINT TESTA LISTING
GLOBAL TXTLIB TESTLIB PROGLIB
ACCESS 200 E
&BEGSTACK
OS.TEST3.STREAM.BETA
&END
FILEDEF INDD1 E DSN ?
FILEDEF INDD2 READER
FILEDEF OUTFILE DISK TEST DATA A1
LOAD TESTA (START
&IF &RETCODE = 100 &GOTO -RET100
&IF &RETCODE = 200 &GOTO -RET200
&EXIT &RETCODE
-RET100 &CONTINUE
    .
    .
    .
     -RET200 &CONTINUE
    .
    .
    .
```

The &CONTROL and &ERROR control statements in the EXEC procedure ensure that if an error occurs during any part of the EXEC, the remainder of the EXEC does not execute, and the execution summary of the EXEC indicates the command that caused the error.

> **Note:** For the FILEDEF command entered with the DSN ? operand, you must stack the response before issuing the FILEDEF command.

In this example, since the OS data set name has more than eight characters, you must use the &BEGSTACK control statement to stack it. If you use the &STACK control statement, the EXEC processor truncates all words to eight characters.

When your program is finished executing, the EXEC special variable &RETCODE indicates the contents of general register 15 at the time the program exited. You can use this value to perform additional steps in your EXEC procedure. Additional steps are indicated in the preceding example by ellipses.

For detailed information on creating EXEC procedures, see Appendix B, "The CMS EXEC Processor" on page B-1.

Executing Members of OS Module Libraries or CMS LOADLIBS

The OS relocating loader allows the user to load a member of a CMS LOADLIB or an OS module library on an OS formatted disk. The OS LINK, LOAD, ATTACH, and XCTL macros are supported. In addition, the OSRUN command (which generates a LINK SVC) is supported to provide for loading and executing members directly from the console.

For macros, the libraries specified in the LOADLIB global list are searched. If the requested member is not found, CMS looks for a TEXT file by that name; then, if still not found, the TXTLIBs specified in the TXTLIB global list are searched for the member name.

For the OSRUN command, the libraries specified in the LOADLIB global list are searched. If the member is not found and the user has a $SYSLIB LOADLIB file, it is searched for the member name. (TEXT and TEXTLIBs are not considered by OSRUN.)

A FILEDEF command must define any OS module libraries from which members are to be loaded. The DDNAME specified must be $SYSLIB. The filename can be any name, but it must correspond to the name stated in the GLOBAL statement; the filetype must be LOADLIB. To define more than one library with the same DDNAME, use the CONCAT option of the FILEDEF command. Any library to be searched (either CMS LOADLIB or OS module library) must be specified in the GLOBAL LOADLIB statement. The data set with the largest block size should be specified first (both in the FILEDEF and in the GLOBAL list). CMS files do not require a file definition, but if used, the file with the largest block size should be specified first. The GLOBAL list determines the order in which the libraries are searched.

The LKED command is used to create a CMS LOADLIB. For example:

```
LKED TESTFILE
```

takes a CMS TEXT file with the filename of TESTFILE and creates a file named TESTFILE LOADLIB. For more information on input to the LKED command refer to the section, "Specifying Input to the LKED command."

The LOADLIB the LKED TESTFILE example creates is an OS simulated PDS named TESTFILE LOADLIB and contains one member named TESTFILE. To execute TESTFILE using the OSRUN command, the GLOBAL command must be used. For example:

```
GLOBAL LOADLIB TESTFILE
```

The OSRUN command causes the TESTFILE member of the TESTFILE loadlib to be loaded, relocated, and executed:

```
OSRUN TESTFILE
```

If the module to be executed resides in an OS module library on an OS formatted disk, the disk must be accessed and the library must be defined (via the FILEDEF command) to make it known to CMS. For example, if SYS1.TESTLIB is a library on an OS disk and contains a member called TEST1, the following would be required to execute TEST1:

```
ACCESS 250 B (where 250 is the address of the OS disk)
FILEDEF $SYSLIB DISK OSLIB LOADLIB B DSN SYS1 TESTLIB
     (DSORG PO RECFM U BLOCK 7294
GLOBAL LOADLIB OSLIB
OSRUN TEST1
```

The LOADLIB files on OS disks can be concatenated with each other and with CMS LOADLIB files by coding the FILEDEF command with the CONCAT option. For example, two OS files and a CMS LOADLIB are searched for TEXT with the following commands:

```
ACCESS 250 B (if 250 is the address of the OS disk)
FILEDEF $SYSLIB DISK OSLIB LOADLIB DSN SYS1 LIB1
     (DSORG PO RECFM U BLOCK 7294)
FILEDEF $SYSLIB DISK MYLIB LOADLIB B DSN SYS1 LIB2 (CONCAT)
GLOBAL LOADLIB OSLIB MYLIB CMSLIB
OSRUN TEST
```

Note: The first FILEDEF command for $SYSLIB must describe the first library filename in the GLOBAL list. Its attribute will be used when the libraries are searched. It is advisable not to code the CONCAT option on the first FILEDEF command so that it clears all previous FILEDEFs for that ddname.

The LOADLIB command provides the utility necessary to maintain the CMS LOADLIBs. The following functions are provided:

COPY	Copy members from one LOADLIB to another Merge complete LOADLIBs Copy with SELECT or EXCLUDE
COMPRESS	Compress a CMS LOADLIB
LIST	LIST members of a CMS LOADLIB

For more detailed information on the LKED, GLOBAL, OSRUN, and LOADLIB commands, refer to the *VM/SP CMS Command and Macro Reference.*

Specifying Input to the LKED Command

Primary LKED input is a data set known to the linkage editor as SYSLIN, which can be described in the FNAME operand of the LKED command. The filetype of the input file named in the command line must be TEXT. Optionally, you can override the FNAME operand by issuing a FILEDEF that defines SYSLIN as the ddname of an alternate primary input source. If your alternate input is a CMS file, the choice of filetype is unrestricted. The contents of the SYSLIN dataset may be:

1. Object text such as assembler or compiler output
2. Linkage editor control statements
3. A combination of object text and control statements.

Linkage editor control statements can be inserted before, between, and after object modules and other control statements. Editing procedures can be used to construct files to meet your requirements. Linkage editor INCLUDE statements may be used to designate explicitly the following files or file members as secondary linkage editor input:

1. CMS TEXT files
2. CMS TEXTLIB files
3. CMS LOADLIB files
4. Members of OS object libraries
5. Members of OS load libraries

A FILEDEF must be issued before the LKED command to define a unique ddname for each file to be included as secondary linkage editor input. An INCLUDE statement in the SYSLIN dataset must specify the ddname assigned to the file by your FILEDEF. For library files, the statement must also specify all

members of the library that are to be included as input. The use of all FILEDEF commands and INCLUDE statements to identify input files is shown in the following examples.

```
CMS commands:

FILEDEF LIBDEF DISK MYLIB TXTLIB B
FILEDEF TXTDEF DISK MYFILE TEXT C

SYSLIN input:

INCLUDE LIBDEF(CSECT1,CSECT2)
INCLUDE TXTDEF
```

INCLUDE statements must begin in column 2. The applicable statement formats are described in the *OS/VS Linkage Editor and Loader.*

Automatic library search is available for either CMS or OS type library members if the FILEDEF for the dataset to be searched specifies SYSLIB as the ddname. Additional libraries can be selected for automatic search by placing linkage editor LIBRARY statements in your SYSLIN input file. Each library statement must contain the associated ddname and a list of members within the library to be included in the search. A FILEDEF must be issued before the LKED command to assign a unique ddname to each dataset to be searched. The library search conducted during a single linkage editor execution is limited to either object-type or load-type modules and may not combine both types. The CONCAT option of the FILEDEF command is not valid for LKED input datasets. To expand the use of the automatic SYSLIB search, the user may combine the menbers of several CMS libraries into a single composite library. The automatic search facility applies to CMS TXTLIBs and LOADLIBs and to OS object libraries and LOAD libraries. The following example shows FILEDEF commands and SYSLIN input for an automatic library search.

```
CMS commands:

FILEDEF SYSLIB DISK SEARCH1 TXTLIB B
FILEDEF LIBDEFA DISK SEARCH2 TXTLIB c
FILEDEF LIBDEF DISK OSTEXT LIBRARY D DSN OBJMODS

SYSLIN input:

LIBRARY LIBDEFA(CSECT1,CSECT2)
LIBRARY LIBDEFB(MEMBER1,MEMBER2)
```

LIBRARY statements must begin in column 2. The GLOBAL command is not needed to identify linkage editor input libraries. For LOADLIB input to the linkage editor, the RECFM U option of the FILEDEF command must be specified.

Chapter 10. Developing VSE Programs Under CMS

You can use CMS to create, compile, execute and debug VSE programs written in assembler, COBOL, PL/I or RPG-II programming languages. CMS simulates many functions of the Disk Operating System VSE so that you can use the interactive facilities of VM/SP to develop your programs, and then execute them in a VSE virtual machine.

This section tells you how to use the CMS/DOS environment. It describes the CMS commands you can use to manipulate DOS disks and DOS files and CMS/DOS commands you can use to simulate the functions of VSE:

- The CMS/DOS environment
- Using DOS files on DOS disks
- Using the ASSGN command
- Using the DLBL command
- Using DOS libraries in CMS/DOS
- Using macro libraries
- VSE assembler language macros supported
- Assembling source programs
- Link-editing programs in CMS/DOS
- Executing programs in CMS/DOS

For a practice terminal session using the commands and techniques presented in this section, see Appendix F, "Sample Terminal Sessions."

A Word About Terminology

CMS/DOS is neither CMS nor is it DOS; it is a composite, and its vocabulary contains both CMS and VSE terms. CMS/DOS performs many of the same functions as DOS, but where, under VSE, a function is initiated by a control card, in CMS it is initiated by a command. Many CMS/DOS commands, therefore, have the same names as the VSE control statement that performs the same function. In those cases where the control statement you would use in VSE and the command you use in CMS are different, the differences are explained. For the most part, whenever a term that is familiar to you as a VSE term is used, it has the same meaning to CMS/DOS, unless otherwise indicated.

CMS/DOS support in VM/SP is based on the VSE program product. The term DOS, however, continues to be used in a general sense and, in the discussion that follows, refers to the VSE program product.

The CMS/DOS Environment

After you have loaded CMS into your virtual machine you can enter the CMS/DOS environment by issuing:

```
set dos on
```

If you want to access a DOS system residence volume during your CMS/DOS terminal session, you should link to and access the disk that contains the DOS SYSRES before you issue the SET command. For example, if you share the system residence volume with other users and it is in your directory at virtual address 390, you would issue the command:

```
access 390 g
```

and then issue the SET command as follows:

```
set dos on g
```

to indicate that the SYSRES is located on your G-disk. If you are going to use the CMS/DOS librarian facilities to access any of the libraries on the system residence volume, you must enter the CMS/DOS environment this way.

If you are using CMS exclusively for DOS applications, you could put the ACCESS and SET DOS ON commands in your PROFILE EXEC.

If you are going to use access method services functions in CMS/DOS, or execute functions that read or write VSAM data sets, you must use the VSAM option of the SET DOS ON command:

```
set dos on g (vsam
```

When you are using CMS/DOS, you can use your virtual machine just as you would if you were in the CMS environment; but you cannot execute any CMS commands or program modules that load and/or use OS macros. The SCRIPT command, for example, uses OS macros, and is therefore invalid in the CMS/DOS environment.

You have, however, in addition to the CP and CMS commands available, a series of commands that simulate VSE functions. Except for the DLBL and DOSLIB commands, these commands or operands should only be issued in the CMS/DOS environment.

The CMS/DOS commands are summarized in Figure 10-1 on page 10-3.

Command	Function
ASSGN	Relates system and programmer logical units to physical devices.
DLBL	Relates a program DDname (filename) to a real disk file so you can perform input/output operations on it.
DOSLIB	Lists or deletes phases from a CMS/DOS phase library, or compresses the library.
DOSLKED	Link-edits CMS TEXT files or DOS phases from system or private relocatable libraries.
DSERV	Displays the directories of DOS libraries.
DOSPLI	An EXEC procedure that invokes the DOS/VS PL/I compiler.
ESERV	An EXEC procedure that invokes the ESERV utility functions on edited assembler language macros.
FCOBOL	An EXEC procedure that invokes the DOS/VS COBOL compiler.
FETCH	Loads executable phases from a DOSLIB or DOS library into storage for execution, and optionally starts execution.
GLOBAL	When you want DOSLIBs searched for executable phases or macro libraries searched for macro definitions, you must identify them with the GLOBAL command.
LISTIO	Displays the current assignments of system and programmer logical units, and optionally creates an EXEC file to contain the information.
OPTION	Sets or changes the options in effect for the DOS/VS COBOL compiler.
QUERY	Use QUERY command operands to list current DLBL definitions (QUERY DLBL), to determine whether or not you are in the CMS/DOS environment (QUERY DOS), the setting of the UPSI byte (QUERY UPSI), the DOSLIBs identified by GLOBAL commands (QUERY DOSLIB or or QUERY LIBRARY), the current number of lines per page (QUERY DOSLNCNT), which options are in effect for the COBOL compiler (QUERY OPTION), or to find out whether you have set a virtual partition size (QUERY DOSPART).
PSERV	Creates CMS files with a filetype of PROC from the VSE procedure library, or displays, prints or punches procedures.
RSERV	Copies a relocatable module from a DOS library and places it in a CMS file with a filetype of TEXT, or displays, prints, or punches modules.
SET	The SET command has operands that allow you to enter or leave the CMS/DOS environment (SET DOS ON or SET DOS OFF) to set the number of SYSLST lines per page (SET DOSLNCNT), to set the UPSI byte (SETUPSI), and to set a virtual partition size (SET DOSPART).
SSERV	Creates CMS COPY files from books on VSE source statement libraries.

Figure 10-1. CMS/DOS Commands and CMS Commands with Special Operands

DL/I in the CMS/DOS Environment

Batch DL/I programs can be written and tested in the CMS/DOS environment.

This includes programs written in assembler, COBOL, and PL/I languages. Not all functions of COBOL and PL/I are supported. For a description of what is supported, see the documentation on the appropriate program product. Data base description generation and program specification block generation can also be executed. However, the application control block generation must be submitted to a DOS virtual machine for execution. The data base recovery and reorganization utilities must also be executed in a DOS virtual machine. This support provides the ability to:

- Interactively code DL/I control blocks and application programs that contain imbedded DL/I calls.
- Store and maintain macros used to generate DL/I control blocks, and programs created under CMS, in the CMS library. Production libraries are thus isolated from the test environment.
- Modify and compile programs using the CMS/DOS text manipulation and EXEC facilities.
- Link-edit and execute batch DL/I programs either interactively or in CMSBATCH. Online DL/I application programs requiring access to CICS/VS must be submitted to a DOS virtual machine for link-editing, cataloging, and execution.

The following restrictions apply:

- All the existing guidelines and restrictions that apply to VSAM data set creation, maintenance, and application program use apply to DL/I data sets.
- The CMS/DOS restriction on writing to sequential files applies to SHSAM and HSAM.
- To assemble a DBD or PSB under CMS/DOS, you must first copy the DBDGEN and PSBGEN macros from the DOS source statement library to a CMS MACLIB.

For more information about using DL/I in the CMS/DOS environment, see *DL/I DOS/VS Data Base Administration.*

Using DOS Files on DOS Disks

You can have DOS disks attached to your virtual machine by a directory entry or you can link to a DOS disk with the LINK command. You can use the ACCESS command to assign a mode letter to the disk:

```
access 155 b
```

and the RELEASE command to release it:

```
release b
```

Except for VSAM disks, you cannot write on DOS disks, or update DOS files on them. You can, however, execute programs and CMS/DOS commands that read from these files, and you can use the LISTDS command to display the fileids of files on a DOS disk; for example:

```
listds b
```

You can also verify the existence of a particular file. For example, if the file-id is NEW.TEST.DATA you can enter:

```
listds new.test.data.b
      -- or --
listds new test data b
```

If the file-id of the DOS file you want to verify contains embedded blanks, for example NEW.TEST DATA, then you have to enter the LISTDS commands with a question mark:

```
listds ? b
```

CMS responds:

```
ENTER DATA SET NAME:
```

and you can enter the exact file-id:

```
new.test data
```

If the data set exists, you receive a response:

```
FM DATA SET NAME
B  NEW.TEST DATA
```

Reading DOS Files

Under CMS/DOS, you can execute programs that read DOS sequential (SAM) files; you can also execute programs that read and write VSAM files. You cannot, however, execute programs to read direct (DAM) or indexed sequential (ISAM) DOS files.

Complete information on using CMS to access and manipulate VSAM files is described in Chapter 11, "Using Access Method Services and VSAM Under CMS and CMS/DOS" on page 11-1. The discussion below lists the restrictions placed on reading SAM files.

Restrictions on Reading DOS Disk Files in CMS

CMS cannot read DOS files that:

- Have the input security indicator on.
- Contain more than 16 user labels and/or data extents. (If the file has user labels, they occupy the first extent; therefore the file must contain no more than 15 data extents.)
- Are multivolume files. Multivolume files are read as single-volume files. End of volume is treated as end of file. There is no end-of-volume switching.
- Have user labels. User labels in user-labeled files are bypassed.

CMS does not support duplicate volume labels; you cannot access more than one volume with the same six-character label while you are using CMS/DOS.

Creating CMS Files from DOS Libraries

You can create CMS files from existing DOS files on DOS disks. CMS simulates the DOS librarian functions DSERV, RSERV, SSERV, ESERV, and PSERV with commands of the same names; you can use these CMS/DOS commands to create CMS files from relocatable, source statement, or procedure libraries located either on the DOS system residence volume or in private libraries. The functions are fully described later in this section.

Copying DOS Disk and Tape Data Files

If you want to create CMS files from DOS files that are not cataloged in libraries or from DOS files on tape, you can use the MOVEFILE command. The MOVEFILE command allows you to copy a file from one device to another device of the same or a different type. Before issuing the MOVEFILE command, the input and the output files must be described to CMS with the FILEDEF command.

The MOVEFILE and FILEDEF commands are described and examples are given of how to use them in Chapter 9, "Developing OS programs under CMS" on page 9-1. The procedures are the same for copying DOS files as for OS data sets. You must, however, keep the following in mind:

- Because DOS files on DOS disks do not contain BLKSIZE, RECFM, or LRECL options, these options must be specified via the FILEDEF command; otherwise, defaults of BLOCKSIZE=32760 and RECFM=U are assigned. LRECL is not used for RECFM=U files.
- If a DOS file-id does not follow OS naming conventions (that is, one- to eight-byte qualifiers with each qualifier separated by a period; up to 44 characters including periods), you must use the DSN ? operand of FILEDEF and the ? operand of LISTDS to enter the DOS file-id.

Copying Modules from VSE Library or SYSIN Tapes

You can create individual CMS files for VSE modules from a VSE library distribution tape or VSE SYSIN tape. Use the VMFDOS command. The VMFDOS command can create a CMS file for each VSE module that exists, and the CMS filename corresponds to the VSE module name. You can restore individual modules, groups of modules, or the entire module set.

For VSE library distribution tapes, the VMFDOS command restores modules from either system or private (relocatable and/or source statement) libraries. The created CMS files have a filetype of 'TEXT' if they are from a relocatable library. They have a filetype of "MACRO" if they are from a source statement library.

For VSE SYSIN tapes, modules containing a period as the second character (for example, "A.") of a VSE "CATALx" control statement have a filetype of 'MACRO'. All other files have a filetype of "TEXT."

The VMFDOS command is described in the *VM/SP Installation Guide*.

Reading in Real Card Decks

If you have DOS files or source programs on cards, you can create CMS files directly by having these cards read into the real system card reader. You direct the cards to your virtual machine by punching a CP ID card in the following format. For example, if your userid is HARMONY, then enter:

```
ID HARMONY
```

and placing this card in front of your card deck. When the cards appear in your virtual card reader, you can read them onto your CMS A-disk with the READCARD command:

```
readcard dataproc assemble
```

You can use the editor to remove any DOS control cards that may be included in the deck.

Using Tapes in CMS/DOS

See Chapter 6, "Using Real Printers, Punches, Readers, and Tapes" for a description of CMS tape label processing for CMS/DOS tape files. The support for tape labels is only for files defined by a DTFMT macro. If you do not use this macro, CMS bypasses IBM standard labels on input tapes and writes a tape mark over any existing labels on an output tape. The CMS LABELDEF command is equivalent in CMS/DOS to the VSE TLBL control statement when standard tape label processing is used.

Using the ASSGN Command

The ASSGN and DLBL commands perform the same functions for CMS/DOS as the ASSGN and DLBL control statements in VSE. You use the ASSGN command to designate an I/O device for a system or programmer logical unit (SYSxxx) and, if the device is a disk device, you can use the DLBL command to establish a real file identification for a symbolic filename in a program. The DLBL command is described under "Using the DLBL Command."

In addition to using the ASSGN command to relate real I/O devices with symbolic units, you must use it in CMS/DOS to:

- Assign SYSIN or SYSIPT for the input source file for a language compiler when you use the DOSPLI or FCOBOL commands.
- Identify the disk, by mode letter, on which a private core image, relocatable, or source statement library resides.
- Assign SYSIN or SYSIPT to the CMS disk on which an ESERV file, containing control statements for the ESERV program, resides.

When you enter the ASSGN command, you must supply the logical unit and the device; for example:

```
assgn sys100 printer
```

assigns the logical unit SYS100 to the printer. When you want to make an assignment to a disk device, you must specify the mode letter at which the disk is accessed. The command:

```
assgn sys010 b
```

assigns the logical unit SYS010 to your B-disk.

The system logical units you can assign and the valid device types you can assign to them in CMS/DOS follow.

SYSIPT, SYSRDR, SYSIN:

These units can be assigned to disk (mode), TAPE, or READER. If you make an assignment to SYSIN, both SYSRDR and SYSIPT are also assigned the same device. Assignment to DOS FB-512 disks is not supported.

SYSLST:

The system logical unit for listings can be assigned to disk (mode), PRINTER, or TAPE.

SYSLOG:

Terminal or operator output or messages can be assigned to PRINTER or TERMINAL. CMS/DOS always assigns SYSLOG to TERMINAL by default, so you never have to make this assignment except when you want to alter it.

SYSPCH:

Punched output, for example text decks, can be assigned to PUNCH, disk (mode), or TAPE.

SYSCLB, SYSRLB, SYSSLB:

The system logical units SYSCLB, SYSRLB, and SYSSLB can be assigned to private core image, relocatable, and source statement libraries, respectively. The only valid assignments for these units is to disk (mode). If you want to reference private libraries with the DOSLKED, DSERV, ESERV, FETCH, SSERV, or RSERV commands, you must assign SYSCLB, SYSRLB, or SYSSLB to the disks on which the libraries reside.

Manipulating Device Assignments

You can assign programmer logical units SYS000 through SYS241 with the ASSIGN command. Besides assigning I/O devices, the ASSGN command can also negate a previous assignment:

```
assgn syspch ua
```

or specify that, for a given device, no real I/O operation is to be performed during the execution of a program:

```
assgn sys009 ign
```

When you release a disk from your virtual machine, any assignments made to that disk are unassigned.

You can find out the current assignments for system and programmer logical units with the LISTIO command, which lists all the system or programmer logical units, even those that are unassigned:

```
listio
```

To list only currently assigned units, enter:

```
listio a
```

To find out the current assignment of one specific unit, for example SYS100, enter:

```
listio sys100
```

With the EXEC option of the LISTIO command, you can create a disk file containing the list of assignments. The $LISTIO EXEC that is created contains two EXEC numeric variables, &1 and &2, for each unit listed. For example, if you entered the command:

```
listio sys081 (exec
```

then the file $LISTIO EXEC may contain the record:

```
&1 &2 SYS081 PRINTER
```

When you use the STAT option, LISTIO lists, for disk devices, whether the disk is read-only or read/write; for example:

```
listio sys100
SYS100 B   R/W
```

indicates that SYS100 is assigned to the B-disk, which is a read/write disk.

You can cancel all current assignments by leaving the CMS/DOS environment and then re-entering it:

```
set dos off
set dos on
```

Virtual Machine Assignments

When you assign a physical device type to a system or programmer logical unit, CMS relates the device to your virtual machine configuration; you receive an error message if you try to assign a logical unit to a device not in your configuration. For example, if you are using the ASSGN command to assign a logical unit to a disk file, you must specify the access mode letter of the disk. If the disk is not accessed, the ASSGN command fails. For another example, if you issue:

```
assgn syspch punch
```

the punch specified is your own virtual machine card punch. The actual destination of punched output then depends on the spooling characteristics of the punch; if it is spooled to another user or to *, then no real cards are punched, but virtual card images are placed in the virtual reader of the destination userid, which may be another virtual machine or your own.

CMS supports only one reader, one punch, and one printer; you cannot make any assignments for multiple output devices in CMS/DOS. When you make an assignment for a logical unit that has already been assigned, it replaces the current assignment.

Using the DLBL Command

Use the DLBL command to supply CMS/DOS with specific file identification information for a disk file that is going to be used for input or output. For any

DLBL command you issue, you must previously have issued an ASSGN command for the disk, specifying a system or programmer logical unit. The basic relationship is:

```
assgn SYSxxx mode
dlbl filename mode DSN ? (SYSxxx
```

Both the SYSxxx and the mode values must match on the ASSGN and DLBL commands; the disk on which the file resides must be accessed at mode.

The filename on the DLBL command line, called a ddname in CMS/DOS, corresponds to the symbolic name for a file in a program. If you want to reference a private DOS library, you must use one of the following ddnames:

System Logical Unit	Filename
SYSCLB	IJSYSCL
SYSRLB	IJSYSRL
SYSSLB	IJSYSSL

Entering File Identifications

When you issue the DLBL command you must identify the file, by file-id (for a VSE file) or by file identifier (for a CMS file). The keywords DSN and CMS indicate whether it is a VSE file or a CMS file, respectively.

If the file is a VSE file residing on a DOS disk, you can enter the DLBL command in one of three ways. For example, for a file named TEST.FILE.INPUT you may enter either:

```
assgn sys101 d
dlbl infile d dsn test.file.input (sys101

     -- or --

dlbl infile d dsn test file input (sys101

     -- or --

assgn sys101 d
dlbl infile d dsn ? (sys101
ENTER DATA SET NAME:
test.file.input
```

For any VSE file with a file-id that contains embedded blanks, you must use the "DSN ?" form.

When you issue a DLBL command for a CMS file, you enter the filename and filetype following the keyword CMS:

```
assgn sys102 a
dlbl outfile a cms new output (sys102
```

In this example, if SYS102 is defined as an output file for a program, the output is written to your CMS A-disk in a file named NEW OUTPUT.

You can, for convenience, use a CMS default file identifier. If you enter:

```
dlbl outfile a cms (sys102
```

then the output filetype defaults to that of the ddname and the filename to FILE. So, this output file is named FILE OUTFILE.

Clearing and Displaying File Definitions

You can clear a DLBL definition for a file by using the CLEAR operand of the DLBL command:

```
dlbl outfile clear
```

To clear all existing definitions, except those entered with the PERM option, you can enter:

```
dlbl * clear
```

This command is issued by the assembler and the language processors when they complete execution. Definitions entered with the PERM option must be individually cleared.

Whenever you use the HX Immediate command to halt the execution of a program, the DLBL definitions in effect are cleared, including those entered with the PERM option.

You can find out what definitions are currently in effect by issuing the DLBL command with no operands:

```
dlbl
```

or, you can use the QUERY command with the DLBL operand.

Using DOS Libraries in CMS/DOS

CMS/DOS provides you with the capability of using various types of files from DOS system or private libraries. You can copy, punch, display at the terminal, or print:

- Books from system or private source statement libraries using the SSERV command
- Relocatable modules from system or private relocatable libraries using the RSERV command
- Procedures from the system procedure library using the PSERV command

You can also:

- Copy and de-edit macros from system and private E sublibraries using the ESERV command
- Access the directories of system or private libraries using the DSERV command
- Link-edit relocatable modules from system or private relocatable libraries with the DOSLKED command
- Read core image phases from system or private core image libraries into storage for execution using the FETCH command

The SSERV Command

If you have cataloged source programs or copy files on the system source statement library and you want to use CMS to modify and test them, you can copy them into CMS files using the SSERV command. For example, suppose you want to copy a book named PROCESS from the A sublibrary on the system residence volume. The DOS system residence is in your virtual machine configuration at virtual address 350, and you have accessed it as your F-disk. First, to indicate to CMS/DOS that the system residence is on your F-disk, you enter:

```
set dos on f
```

then you can enter the SSERV command, specifying the sublibrary identification and the book name:

```
sserv a process
```

This creates, from the A sublibrary, a file named PROCESS COPY and places it on your A-disk. If the book contained assembler language source statements you would want the filetype to be ASSEMBLE, so you may enter:

```
sserv a process assemble
```

If you want to copy a book from a private source statement library, you must first use the ASSGN and DLBL commands to make the library known to CMS/DOS. For example, to obtain a copy file from a private library on a DOS disk accessed as your D-disk, enter:

```
assgn syssslb d
dlbl ijsyssl d dsn ? (syssslb
ENTER DATA SET NAME:
program.test library
```

Now, when you enter the SSERV command:

```
sserv t setup copy
```

the book named SETUP in the T sublibrary of PROGRAM.TEST LIBRARY is copied into a CMS file named SETUP COPY. If SETUP is not found in the private library, then CMS searches the system library, if it is available.

The RSERV Command

In CMS/DOS, to manipulate relocatable modules that have been cataloged either on the system or a private relocatable library you must first copy them into CMS files with the RSERV command. You can link-edit modules directly from DOS relocatable libraries, but if you want to add or modify linkage editor control statements for a module, you must place the control statements in a CMS file.

If you are copying a relocatable module from the system relocatable library, then you should make sure that you have indicated the system residence disk when you entered the CMS/DOS environment:

```
set dos on f
```

then you can issue the RSERV command specifying the name of the relocatable module you want to copy:

```
rserv rtna
```

The execution of this command results in the creation of a CMS file named RTNA TEXT on your A-disk.

If you want to copy a relocatable module from a private relocatable library, you must first use the ASSGN and DLBL commands to make the private library known to CMS/DOS:

```
assgn sysrlb d
dlbl ijsysrl d dsn reloc.lib (sysrlb
```

Then, issue the RSERV command for a specific module in that library:

```
rserv testrtna
```

to create the CMS file TESTRTNA TEXT from the module named TESTRTNA. If the module TESTRTNA is not found in RELOC.LIB, CMS searches the system library, if it is available.

The PSERV Command

If you want to copy DOS cataloged procedures into CMS files to use, for example, in preparing job streams for a DOS virtual machine, you can use the PSERV command:

```
pserv prepjob
```

This command creates a CMS file on your A-disk; the file is named PREPJOB PROC. To copy a procedure from the procedure library you must have entered the CMS/DOS environment specifying a disk mode for the system residence volume.

You cannot execute DOS/VS procedures directly from the CMS/DOS environment. However, if you modify a procedure, you can punch it to a virtual machine that is running a DOS system, and execute it there.

The ESERV Command

The CMS/DOS ESERV command is actually an EXEC procedure that calls the VSE ESERV utility program. To use the ESERV program, you first must IPL CMS with a CMSBAM DCSS (shared segment), then create a file with a filetype of ESERV that contains the ESERV control statements you want to execute. For example, if you want to write a de-edited copy of the macro DTFCD onto your A-disk, you might create a file named DTFCD ESERV, with the record:

```
PUNCH E.DTFCD
```

As when you submit ESERV jobs in DOS column 1 must be blank.

Prior to executing the ESERV program, you must enter the CMS/DOS environment by specifying the SET DOS ON command using a VSE system residence volume. This is necessary because the ESERVE procedure invokes the ESERV program directly from the VSE core image library.

Then, you must assign SYSIN to the device on which the ESERV source file resides, usually your A-disk:

```
assgn sysin a
```

Then you can enter the ESERV command specifying the filename of the ESERV file:

```
eserv dtfcd
```

No other ASSGN commands are required; the CMS/DOS ESERV EXEC makes default assignments for SYSPCH and SYSLST to disk.

To copy and de-edit macros from a private E sublibrary, issue the ASSGN and DLBL commands to identify the library. For example, to identify a source statement library named TEST.MACROS on the DOS disk accessed as the C-disk, enter:

```
assgn syssslb c
dlbl ijsyssl c dsn test.macros (syssslb
```

The SYSLST output is contained in a CMS file with the same filename as the ESERV file and a filetype of LISTING; you must examine the LISTING file to see if the ESERV program executed successfully. You can either edit it, or display its contents with the TYPE command:

```
type dtfcd listing
```

The SYSPCH output is contained in a file with the same name as the ESERV file and a filetype of MACRO. If you want to punch ESERV output to your virtual card punch, make an assignment of SYSPCH to PUNCH.

When you use the PUNCH or DSPCH ESERV control statements, CATAL.S, END, or /* records may be inserted in the output file. When you use the MACLIB command to add the MACRO file to a CMS macro library, these statements are ignored.

See "Using Macro Libraries" for information on creating and manipulating CMS macro libraries.

The DSERV Command

You can use the DSERV command to examine the contents of system or private libraries. If you do not specify any options with it, the DSERV command creates a disk file, named DSERV MAP, on your A-disk. You can use the PRINT or TERM options to specify that the directory list is either to be printed on your spooled printer or displayed at your terminal. You can also use the SORT option to create a list in collating sequence.

In order to examine a system directory, you must have entered the CMS/DOS environment specifying the mode letter of the DOS system residence:

```
set dos on f
```

If you want to examine the directory of a private source statement, core image, or relocatable library you must issue the ASSGN and DLBL commands establishing SYSSLB, SYSCLB, or SYSRLB before using the DSERV command.

For example, to display at your terminal an alphameric list of procedures cataloged on the system procedure library, you would issue:

```
dserv pd (sort term
```

If the directory you are examining is for a core image library, you can specify a particular phase name to ascertain the existence of the phase:

```
dserv cd phase $$bopen (term
```

To list the directory of a private source statement library, you would first issue the ASSGN and DLBL commands:

```
assgn syssslb b
dlbl ijsyssl b dsn test.source (syssslb
```

then enter the DSERV command:

```
dserv sd
```

The CMS file, DSERV MAP A, that is created in this example contains the directory of the private source statement library TEST.SOURCE.

Using DOS Core Image Libraries

You can load core image phases from DOS core image libraries into virtual storage and execute them under CMS/DOS. Since CMS cannot write directly to DOS disks, linkage editor output under CMS/DOS is placed in a special CMS file called a DOSLIB. When you execute the FETCH command in CMS/DOS you can load phases from either system or private DOS core image libraries as well as from CMS DOSLIBs. More information on using the FETCH command is contained under "Executing Programs in CMS/DOS."

Using Macro Libraries

DOS macro libraries cannot be accessed directly by the VM/SP assembler. If you want to assemble DOS programs in CMS/DOS that use DOS macro or copy files that are on the system or a private macro library you must first create a CMS macro library (MACLIB) containing the macros you wish to use. Since the process of creating a CMS MACLIB from the DOS system source statement library (E sublibrary) can be very time-consuming, you should check with your installation's system programmer to see if it has already been done, and to verify the filename of the macro library, so that you can use it in CMS/DOS.

> **Note:** The DOS, PL/I and DOS/VS COBOL compilers executing in CMS/DOS cannot read macro or copy files from CMS MACLIBs. Macros and copy files are obtained instead from a DOS source statement library.

If you want to extract DOS system macros to modify them for your private use, or if you want to use macros from a private library in CMS, you must use the procedure outlined below to create the MACLIB files.

CMS MACLIBs

A CMS macro library has a filetype of MACLIB. You can create a MACLIB from files with filetypes of MACRO or COPY. A MACRO file may contain macro definitions; COPY files contain predefined source statements.

When you want to assemble a source program that uses macro or copy definitions, you must ensure that the library containing the code is identified before you invoke the assembler. Otherwise, the library is not searched. You identify libraries to be searched using the GLOBAL command. For example, if you have two MACLIBs that contain your private macros and copy files whose names are TESTMAC MACLIB and TESTCOPY MACLIB, you would issue the command:

```
global maclib testmac testcopy
```

The libraries you specify on a GLOBAL command line are searched in the order you specify them. A GLOBAL command remains in effect for the remainder of your terminal session, or until you IPL CMS. To find out what macro libraries are currently available for searching, issue the command:

```
query maclib
```

You can reset the libraries or the search order by reissuing the GLOBAL command.

Creating a CMS MACLIB

To create a CMS macro library, each macro or copy file you want included in the MACLIB must first be contained in a CMS file with a filetype of COPY or MACRO. If you are creating a CMS MACLIB file from a DOS library you must use the SSERV command to copy a file from any source statement library other than an E sublibrary, or use the ESERV command to copy and de-edit a macro from an E sublibrary. The SSERV command uses a default filetype of COPY; the ESERV command uses a default filetype of MACRO.

The following example shows how to copy macros from various sources and shows how to create and use the CMS MACLIB that contains these macros.

1. Enter the CMS/DOS environment with the DOS system residence on a disk accessed as mode C:

   ```
   set dos on c
   ```

2. Copy the macro book named OPEN from the A sublibrary of the system source statement library:

   ```
   sserv a open
   ```

3. Establish a private source statement library:

   ```
   access 351 d
   assgn sysslb d
   dlbl ijsyssl d dsn ? (sysslb
   test source.lib
   ```

4. Issue the SSERV command for a macro in the M sublibrary of TEST SOURCE.LIB:

   ```
   sserv m releas
   ```

5. Create an ESERV file to copy from the E sublibrary:

   ```
   xedit contrl eserv
   input punch contrl
   file
   ```

6. Execute the ESERV command:

   ```
   assgn sysin a
   eserv contrl
   ```

7. Create a CMS macro library named MYDOSMAC from the files just created, which are named OPEN COPY, RELEAS COPY, and CONTRL MACRO:

   ```
   maclib gen mydosmac open releas contrl
   ```

8. To use these macros in an assembler language program, you must indicate that this MACLIB is accessible before assembling a source file:

```
global maclib mydosmac
```

The MACLIB Command

The MACLIB command performs a variety of functions. You use it to:

- Create the MACLIB (GEN function)
- Add, delete, or replace members (ADD, DEL, and REP functions)
- Compress the MACLIB (COMP function)
- List the contents of the MACLIB (MAP function)

Descriptions of these MACLIB command functions follow.

GEN Function:

The GEN (generate) function creates a CMS macro library from input files specified on the command line. The input files must have filetypes of either MACRO or COPY. For example:

```
maclib gen mymac get pdump put regequ
```

creates a macro library with the file identifier MYMAC MACLIB A1 from macros existing in the files with the file identifiers:

```
GET {MACRC}, PDUMP {MACRO}, PUT {MACRO}, and REGEQU {MACRO}
    {COPY }        {COPY }      {COPY }              {COPY }
```

If a file named MYMAC MACLIB A1 already exists, it is erased.

Assume that the files GET MACRO, PDUMP COPY, PUT MACRO, and REGEQU COPY exist and contain macros in the following form:

```
GET MACRO     PDUMP COPY      PUT MACRO     REGEQU COPY
   GET        *COPY PDUMP     PUT           XREG
               PDUMP
   WAIT       *COPY WAIT                    YREG
               WAIT
```

The resulting file, MYMAC MACLIB A1, contains the members:

```
GET             WAIT
WAIT            PUT
PDUMP           REGEQU
```

The WAIT macro, which appears twice in the input to the command, also appears twice in the output. The MACLIB command does not check for duplicate macro names. If, at a later time, the WAIT macro is requested from MYMAC MACLIB, the first WAIT macro encountered in the directory is used.

When COPY files are added to MACLIBs, the name of the library member is taken from the name of the COPY file, or from the *COPY statement, as in the file PDUMP COPY, above.

> **Note** Although the file REGEQU COPY contained two macros, they were both included in the MACLIB with the name REGEQU. When the input file is a MACRO file, the member name is taken from the macro prototype statement in the MACRO file.

ADD Function:

The ADD function appends new members to an existing macro library. For example, assume that MYMAC MACLIB A1 exists as created in the example in the explanation of the GEN function and the file DTFDI COPY exists as follows:

```
*COPY DTFDI
 DTFDI macro definition
*COPY DIMOD
 DIMOD macro definition
```

If you issue the command:

```
maclib add mymac dtfdi
```

the resulting MYMAC MACLIB A1 contains the members:

```
GET             PUT
WAIT            REGEQU
PDUMP           DTFDI
WAIT            DIMOD
```

REP Function:

The REP (replace) function deletes the directory entry for the macro definition in the files specified. It then appends new macro definitions to the macro library and creates new directory entries. For example, assume that a macro library TESTMAC MACLIB contains the members A, B, and C, and that the following command is entered:

```
maclib rep testmac a c
```

The files represented by file identifiers A MACRO and C MACRO each have one macro definition. After execution of the command, TESTMAC MACLIB contains members with the same names as before, but the contents of A and C are different.

DEL Function:: The DEL (delete) function removes the specified macro name from the macro library directory and compresses the directory so there are no unused entries. The macro definition still occupies space in the library, but since no directory entry exists, it cannot be accessed or retrieved. If you attempt to delete a macro for which two macro definitions exist in the macro library, only the first one encountered is deleted. For example:

```
maclib del mymac get put wait dtfdi
```

deletes macro names GET, PUT, WAIT, and DTFDI from the directory of the macro library named MYMAC MACLIB. Assume that MYMAC exists as in the ADD function example. After the above command, MYMAC MACLIB contains the following members:

```
PDUMP
WAIT
REGEQU
DIMOD
```

COMP Function:

Execution of a MACLIB command with the DEL or REP functions can leave unused space within a macro library. The COMP (compress) function removes any macros that do not have directory entries. This function uses a temporary file named MACLIB CMSUT1. For example, the command:

```
maclib comp mymac
```

compresses the library MYMAC MACLIB.

MAP Function:

The MAP function creates a list containing the name of each macro in the directory, the size of the macro, and its position within the macro library. If you want to display a list of the members of a MACLIB at the terminal, enter the command:

```
maclib map mymac (term
```

The default option, DISK, creates a file on your A-disk which has a filetype of MAP and a filename equal to the filename of the MACLIB. If you specify the PRINT option, then a copy of the map file is spooled to your virtual printer as well as being written onto disk.

Manipulating MACLIB Members

The following CMS commands supply a MEMBER option, which allows you to reference individual members of a MACLIB:

- PRINT (to print a member)
- PUNCH (to punch a member)
- TYPE (to display a member)
- FILEDEF (to establish a file definition for a member)

You can use the CMS editor to create the MACRO and COPY files and then use the MACLIB command to place them in a library. Once they are in a library, you can erase the original files.

To extract a member from a macro library, you can use either the PUNCH or the MOVEFILE command. If you use the PUNCH command you can spool your virtual card punch to your own virtual reader:

```
cp spool punch to *
```

Then punch the member:

```
punch testmac maclib (member get noheader
```

and read it back onto disk:

```
readcard get macro
```

In the above example, the member was punched with the NOHEADER option of the PUNCH command, so that a name could be assigned on the READCARD command line. If a header had been created for the file, it would have indicated the filename and filetype as GET MEMBER.

If you use the MOVEFILE command, you must issue a file definition for the input member name and the output macro or copy file before entering the MOVEFILE command:

```
filedef inmove disk testcopy maclib (member enter
filedef outmove disk enter copy a
movefile
```

This example copies the member ENTER from the macro library TESTCOPY MACLIB A into a CMS file named ENTER COPY.

When you use the PUNCH or MOVEFILE commands to extract members from CMS MACLIBs, each member is followed by a // record, which is a MACLIB delimiter. You can edit the file and use the DELETE subcommand to delete the // record.

If you wish to move the complete MACLIB to another file, use the COPYFILE command.

System MACLIBs

The macro libraries that are on the system disk contain CMS and OS assembler language macros. The MACLIBs are:

- CMSLIB MACLIB, which contains the CMS macros from VM/370.
- DMSSP MACLIB, which contains macros that are new or changed in VM/SP.

  Note: When assembling programs that use CMS macros, both of these libraries should be identified via the GLOBAL command. DMSSP should precede CMSLIB in the search order.
- DOSMACRO MACLIB, which contains macros used internally by CMS/DOS

  Note: These macros should not be used in user written programs. To assemble programs that use VSE macros, you should follow the procedures as previously described in this chapter.
- OSMACRO MACLIB, OSMACRO1 MACLIB, and TSOMAC MACLIB, which are used by OS programmers.
- DMKSP MACLIB, which contains macros that support CP.
- OSVSAM MACLIB, which contains the subset of supported OS/VSAM macros.

When you use VSAM on CMS and write programs using VSE/VSAM macros, you can build a VSE/VSAM maclib by issuing the CMS VSEVSAM command. The maclib will contain the supported VSE/VSAM macros and the following VSE macros:

```
CDLOAD
CLOSE
CLOSER
GET
OPEN
OPENR
PUT
```

Refer to the *VM/SP Installation Guide* for the CMS VSEVSAM command documentation.

VSE Assembler Language Macros Supported

Figure 10-2 on page 10-22 lists the VSE assembler language macros supported by CMS/DOS. You can assemble source programs that contain these macros under CMS/DOS, provided that you have the macros available in either your own or a shared CMS macro library. The macros whose functions are described in the "Function" column with the term "no-op" are supported for assembly only; when you execute programs that contain these macros, the VSE functions are not performed. To accomplish the macro function you must execute the program in a VSE virtual machine.

Macro	SVC	Function
CALL		Pass control to another program
CANCEL	06	Terminate processing
CDLOAD	65	Load a VSAM phase
CHECK		Verify completion of a read or write operation
CLOSE/ CLOSER		Deactivate a data file
CNTRL		Control a physical device
COMRG	33	Return address of background partition communication region
DEQ	41	no-op
DTFxx		Establish file definitions
DUMP		Dump storage and registers and terminate processing
ENQ	42	no-op
EOJ	14	Terminate processing normally
ERET		Provide an error routine
EXCP	00	Execute a channel program
EXIT PC	17	Return from program check routine
EXIT AB	95	Return from abnormal termination routine
EXTRACT	98	Retrieve PUB, storage boundaries, or CPUID information
FCEPGOUT	86	no-op
FETCH	01	Load and pass control to a phase
	02	Load and pass control to a logical transient
FREE	36	no-op
FREEVIS	62	Release user free storage
GENL		Generate a phase directory list
GET		Access a sequential file
GETFLD/ MODFLD	107	Provide macro interface support for system information retrieval.
GETVCE	99	Return requested device information to output area.
GETVIS	61	Obtain user free storage
GETIME	34	Get the time of day
JDUMP		Dump storage and registers and terminate processing
LOAD	04	Load a phase into storage
LOCK/ UNLOCK	110	Resource control
MVCOM	05	Modify bytes in the partition communication region
NOTE		Manage data set access
OPEN/ OPENR		Activate a data file

Figure 10-2 (Part 1 of 2). VSE Macros Supported by CMS

Macro	SVC	Function
PAGEIN	87	no-op
PDUMP		Dump storage and registers and continue processing
PFIX	67	no-op
PFREE	68	no-op
POINTR		Position a file for reading
POINTS		Reposition a file to its beginning
POINTW		Position a file for writing
POST	40	Post the event control block
PRTOV		Control printer overflow
PUT		Write to a sequential file
PUTR		Communicate with the system operator
READ		Access a sequential file
RELPAG	85	no-op
RELSE		Skip to begin reading next block
RETURN		Return control to calling program
RUNMODE	66	Check if program is running real or virtual
SECTVAL	75	Obtain a sector number
SETIME	10/24	no-op
SETPFA	71	no-op
STXIT AB	37	Provide or terminate linkage to abnormal ending routine
STXIT PC	16	Provide or terminate linkage to program check routine
STXIT IT	20	no-op
STXIT OC	18	no-op
SUBSID	105	Retrieve information on supervisor subsystem
TRUNC		Skip to begin writing next block
TTIMER	52	Return a 0 in Register 0 (effectively a no-op)
WAIT	07	Wait for the completion of I/O
WRITE		Write to a sequential file
xxMOD		Create Logical IOCS routine inline

Figure 10-2 (Part 2 of 2). VSE Macros Supported by CMS

Assembling Source Programs

If you are a DOS assembler language programmer using CMS/DOS, you should be aware that the assembler used is the VM/SP assembler, not the DOS assembler. The major difference is that the VM/SP assembler, invoked by the ASSEMBLE command, is designed for interactive use, so that when you assemble a program, error messages are displayed at your terminal when compilation is completed, and you do not have to wait for a printed listing to see the results. You can correct your source file and reassemble it immediately. When your program assembles without errors, you can print the listing.

To specify options to be used during the assembly, you enter them on the ASSEMBLE command line. So, for example, if you do not want the output LISTING file placed on disk, you can direct it to the printer:

```
assemble myfile (print
```

All of the ASSEMBLE command options are listed in *VM/SP CMS Command and Macro Reference.*

When you invoke the ASSEMBLE command specifying a file with a filetype of ASSEMBLE, CMS searches all of your accessed disks, using the standard search order, until it locates the file. When the assembler creates the output LISTING and TEXT files, it writes them onto disk according to the following priorities:

1. If the source file is on a read/write disk, the TEXT and LISTING files are written onto the same disk.

2. If the source file is on a read-only disk that is an extension of a read/write disk, the TEXT and LISTING files are written onto the parent disk.

3. If the source is on any other read-only disk, the TEXT and LISTING files are written onto the A-disk.

In all of the above cases, the filenames assigned to the TEXT and LISTING files are the same as the filename of the input file.

The output files used by the assembler are defined via FILEDEF commands issued by CMS when it calls the assembler. If you issue a FILEDEF command using one of the assembler ddnames before you issue the ASSEMBLE command, you can override the default file definitions.

The ddname for the source input file is ASSEMBLE. If you enter:

```
filedef assemble reader
assemble sample
```

then the assembler reads your input file from your card reader, and assigns the filename SAMPLE to the output TEXT and LISTING files. You can use this method to assemble programs directly from DOS sequential files on DOS disks. For example, to assemble a source file named DOSPROG from a DOS disk accessed as a C-disk, you could enter:

```
filedef assemble c dsn dosprog (recfm f lrecl 80
assemble dosprog
```

Again, the name you assign on the ASSEMBLE command may be anything; the assembler uses this name to assign filenames to the TEXT and LISTING output files.

LISTING and TEXT are the ddnames assigned to the SYSLST and SYSPCH output of the assembler. You might issue file definitions to override these defaults as follows:

```
filedef listing disk assemble listfile a
filedef text disk assemble textfile a
assemble source
```

When these commands are executed, the output from the assembly of the file SOURCE ASSEMBLE is written to the disk files ASSEMBLE LISTFILE and ASSEMBLE TEXTFILE.

Link-editing Programs in CMS/DOS

When the assembler or one of the language compilers executes, the object module produced is written to a CMS disk in a file with a filetype of TEXT. The filename is always the same as that of the input source file. These TEXT files (sometimes referred to as decks, although they are not real card decks) can be used as input to the linkage editor or can be the target of an INCLUDE linkage editor control statement.

You can invoke the CMS/DOS linkage editor with the DOSLKED command, for example:

```
doslked test testlib
```

where TEST is the filename of either a DOSLNK or TEXT file (that is, a file with a filetype of either DOSLNK or TEXT) or the name of a relocatable module in a system or private relocatable library. TESTLIB indicates the name of the output file which, in CMS/DOS, is a phase library with a filetype of DOSLIB.

When you issue the DOSLKED command, CMS first searches for a file with the specified name and a filetype of DOSLNK. If none are found, it searches the private relocatable library, if you have assigned one (you must issue an ASSGN command for SYSRLB and use the ddname IJSSYRL in a DLBL statement). If the module is still not found, CMS searches all of your accessed disks for a file with the specified name and a filetype of TEXT. Last, CMS searches the system relocatable library, if it is available (you must enter the CMS/DOS environment specifying the mode letter of the DOS system residence if you want to access the system libraries).

Linkage Editor Input

You can place the linkage editor control statements ACTION, PHASE, INCLUDE, and ENTRY in a CMS file with a filetype of DOSLNK. When you use the INCLUDE statement, you may specify the filename of a CMS TEXT file or the name of a module in a DOS relocatable library:

```
INCLUDE XYZ
```

or you may use the INCLUDE control statement to indicate that the object code follows:

```
INCLUDE
(CMS TEXT file)
```

A typical DOSLNK file, named CONTROL DOSLNK, might contain the following:

```
ACTION REL
PHASE PROGMAIN,S
INCLUDE SUBA
PHASE PROGA,*
INCLUDE SUBB
```

When you issue the command:

```
doslked control
```

the linkage editor searches the following for the object files SUBA and SUBB:

- A DOS private relocatable library, provided you have issued the ASSGN and DLBL commands to identify it:

  ```
  assgn sysrlb d
  dlbl ijsysrl d dsn ? (sysrlb
  ```

- Your CMS disks for files with filenames SUBA and SUBB and a filetype of TEXT

- The system relocatable library located on the DOS system residence volume (if it is available)

Link-editing TEXT Files

When you want to link-edit individual CMS TEXT files, you can insert linkage editor control statements in the file using the editor and then issue the DOSLKED command:

```
xedit rtnb text
input include rtnc
file
doslked rtnb mydoslib
```

When the above DOSLKED command is executed, the CMS file RTNB TEXT is used as linkage editor input, as long as there is no file named RTNB DOSLNK. The ACTION statement, however, is not recognized in TEXT files.

You can also link-edit relocatable modules directly from a DOS system or private relocatable library, provided that you have identified the library. If you do this, however, you cannot provide control statements for the linkage editor.

To link-edit a relocatable module from a DOS private library and add linkage editor control statements to it, you could use this procedure:

1. Identify the library and use the RSERV command to copy the relocatable module into a CMS TEXT file. In this example, the module RTNC is to be copied from the library OBJ.MODS:

   ```
   assgn sysrlb e
   dlbl ijsysrl e dsn obj mods (sysrlb
   rserv rtnc
   ```

2. Create a DOSLNK file, insert the linkage editor control statements, and copy the TEXT file created in step 1 into it using the GETFILE subcommand:

   ```
   xedit rtnc doslnk
   input action rel
   getfile rtnc text a
   file
   ```

3. Invoke the linkage editor with the DOSLKED command:

   ```
   doslked rtnc mydoslib
   ```

Alternatively, you could create a DOSLNK file with the following records:
DOSLNK file

```
ACTION REL
INCLUDE RTNC
```

and link-edit the module directly from the relocatable library. If you do not need a copy of the module on a CMS disk, you might want to use this method to conserve disk space.

When the linkage editor is reading modules, it may encounter a blank card at the end of a file, or a * (comment) card at the beginning of a file. In either case, it issues a warning message indicating an invalid card, but continues to complete the link-edit.

Linkage Editor Output: CMS DOSLIBs

The CMS/DOS linkage editor always places the link-edited executable phase in a CMS library with a filetype of DOSLIB. You should specify the filename of the DOSLIB when you enter the DOSLKED command:

```
doslked prog0 templib
```

where PROG0 is the relocatable module you are link-editing and TEMPLIB is the filename of the DOSLIB.

If you do not specify the name of a DOSLIB, the output is placed in a DOSLIB that has the same name as the DOSLNK or TEXT file being link-edited. In the above example, a CMS DOSLIB is created named TEMPLIB DOSLIB, or, if the file TEMPLIB DOSLIB already exists, the phase PROG0 is added to it.

DOSLIBs can contain relocatable core image phases suitable for execution in CMS/DOS. Before you can access phases in it, you must identify it to CMS with the GLOBAL command:

```
global doslib templib permlib
```

When CMS is searching for executable phases, it searches all DOSLIBs specified on the last GLOBAL DOSLIB command line. If you have named a number of DOSLIBs, or if any particular DOSLIB is very large, the time required for CMS to fetch and execute the phase increases. You should use separate DOSLIBs for executable phases, whenever possible, and then specify only the DOSLIBs you need on the GLOBAL command.

When you link-edit a module into a DOSLIB that already contains a phase with the same name, the directory entry is updated to point to the new phase. However, the space that was occupied by the old phase is not reclaimed. You should periodically issue the command:

```
doslib comp libname
```

where libname is the filename of the DOSLIB, to compress the DOSLIB and delete unused space.

Linkage Editor Maps

The DOSLKED command also produces a linkage editor map, which it writes into a CMS file with a filename that is that of the name specified on the DOSLKED command line and a filetype of MAP. The filemode is always A5. If you do not want a linkage editor map, use the NOMAP option on the ACTION statement in a DOSLNK file.

Executing Programs in CMS/DOS

After you have assembled or compiled a source program and link-edited the TEXT files, you can execute the phases in your CMS virtual machine. You may not, however, be able to execute all your DOS programs directly in CMS. There are a number of execution-time restrictions placed on your virtual machine by VM/SP. You cannot execute a program that uses:

- Multitasking
- More than one partition
- Teleprocessing
- ISAM macros to read or write files
- CMS module files created by DOS programs
- Sets the EC mode bit in the PSW

The above is only a partial list, representing those restrictions with which you might be concerned. For a complete list of restrictions, see the *VM/SP Planning Guide and Reference*. See also the usage notes of the FETCH command in the *VM/SP CMS Command and Macro Reference*.

Executing DOS Phases

You can load executable phases into your CMS virtual machine using the FETCH command. Phases must be link-edited before you load them; they must have been link-edited with ACTION REL. When you issue the FETCH command, you specify the name of the phase to be loaded:

```
fetch myprog
```

Then you can begin executing the program by issuing the START command:

```
start
```

Or, you can fetch a phase and begin executing it on a single command line:

```
fetch prog2 (start
```

When you use the FETCH command without the START option, CMS issues a message telling you at what virtual storage address the phase is loaded:

```
PHASE PROG2 ENTRY POINT AT LOCATION 020000
```

Location X'20000'is the starting address of the user program area for CMS; relocatable phases are always loaded starting at this address unless you specify a different address using the ORIGIN option of the FETCH command:

```
fetch prog3 (origin 22000
start
```

The program PROG3 executes beginning at location 22000 in the CMS user program area.

Search Order for Executable Phases

When you execute the FETCH command, CMS searches for the phase name you specify in the following places:

1. In a DOS private core image library on a DOS disk. If you have a private library you want searched for phases, you must identify it using the ASSGN and DLBL commands, using the logical unit SYSCLB:

   ```
   assgn sysclb d
   dlbl ijsyscl d dsn ? (sysclb
   ```

2. In CMS DOSLIBs on CMS disks. If you want DOSLIBs searched for phases, you must use the GLOBAL command to identify them to CMS/DOS:

   ```
   global doslib templib mylib
   ```

   You can specify up to eight DOSLIBs on the GLOBAL command line.

3. On the DOS system residence core image library. If you want the system core image library searched you must have entered the CMS/DOS environment specifying the mode letter of the system residence:

   ```
   set dos on z
   ```

When you want to fetch a core image phase that has copies in both the core image library and a DOSLIB, and you want to fetch the copy from the CMS DOSLIB, you can bypass the core image library by entering the command:

```
assgn sysclb ua
```

When you need to use the core image library, enter:

```
assgn sysclb c
```

where C is the mode letter of the system residence volume. You do not need to reissue the DLBL command to identify the library.

Making I/O Device Assignments

If you are executing a program that performs I/O, you can use the ASSGN command to relate a system or programmer logical unit to a real I/O device:

```
assgn syslst printer
assgn sys052 reader
```

In this example, your program is going to read input data from your virtual card reader; the output print file is directed to your virtual printer. If you want to reassign these units to different devices, you must be sure that the files have been defined as device independent.

If you assign a logical unit to a disk, you should identify the file by using the DLBL command. On the DLBL command, you must always relate the DLBL to the system or programmer logical unit previously specified in an ASSGN command:

```
assgn sys015 b
dlbl myfile b dsn ? (sys015
```

When you enter the DLBL command with the ? operand you are prompted to enter the DOS file-id.

You must issue all of the ASSGN and DLBL commands necessary for your program's I/O before you issue the FETCH command to load the program phase and begin executing.

Specifying a Virtual Partition Size

For most of the programs that you execute in CMS, you do not need to specify how large a partition you want those programs to execute in. When you issue the START command or use the START option on the FETCH command, CMS calculates how much storage is available in your virtual machine and sets a partition size. CMS calculates how much storage is available in the following manner:

```
FREELOWE - (MAINHIGH + (4096 * FRERESPG))
```

where:

FREELOWE	equals the low extent of allocated storage obtained from the top of virtual storage downwards via the DMSFREE system request.
MAINHIGH	equals the high extent of allocated storage obtained from the low virtual storage upwards via the GETMAIN user request for storage.
FRERESPG	equals the amount of storage to be reserved for subsequent system requests, in pages.

In some instances, you may want to control the partition size:

- For performance considerations
- Because the default may not leave enough free storage to satisfy the GETVIS commands issued by the DOS program or the access method services function being executed.

You can set the partition size with the DOSPART operand of the SET command. For example, after you enter the command:

```
set dospart 300k
```

all programs that you subsequently execute during this session will execute in a 300K partition. In this way you can:

- Set a smaller partition size for programs that run better in smaller partitions.
- Set a smaller partition size to leave more free storage. If the reduction of the DOS partition does not free enough storage for the GETVIS commands, a larger virtual machine must be defined. If you enter:

  ```
  set dospart off
  ```

the CMS calculates a partition size when you execute a program. This is the default setting.

> **Note:** The CMS partition, unlike the DOS partition, is used only for the loading and executing of programs invoked by the FETCH or LOAD commands. Areas allocated by GETVIS will be assigned addresses outside the partition but within the user's virtual machine.

Setting the UPSI Byte

If your program uses the user program switch indicator (UPSI) byte, you can set it by using the UPSI operand of the CMS SET command. The UPSI byte is initially binary zeros. To set it to ones, enter:

```
set upsi 11111111
```

To reset it to zeros, enter:

```
set upsi off
```

Any value you set remains in effect for the duration of your terminal session, unless you reload CMS (with the IPL command).

Debugging Programs in CMS/DOS

You can debug your DOS programs in CMS/DOS using the facilities of CP and CMS. By executing your programs interactively, you can more quickly determine the cause of an error or program abend, correct it, and attempt to execute a program again.

The CP and CMS debugging facilities are described in Chapter 13, "Debugging Your Program Using VM/SP" on page 13-1. Additional information for assembler language programmers is in Chapter 8, "Programming for The CMS Environment" on page 8-1.

Using CMS EXEC Procedures in CMS/DOS

During your program development and testing cycle, you may want to create CMS EXEC procedures to contain sequences of CMS commands that you execute frequently. For example, if you need a number of MACLIBs, DOSLIBs, and DLBL definitions to execute a particular program, you might have an EXEC procedure as follows:

```
&CONTROL ERROR TIME
&ERROR &EXIT &RETCODE
GLOBAL MACLIB TESTLIB DOSMAC
ASSEMBLE TESTA
PRINT TESTA LISTING
DOSLKED TESTA TESTLIB
GLOBAL DOSLIB TESTLIB PROGLIB
ACCESS 200 E
ASSGN SYS010 E
&BEGSTACK
DOS.TEST3.STREAM.BETA
&END
DLBL DISK1 E DSN ? (SYS010
ASSGN SYS011 PUNCH
CP SPOOL PUNCH TO *
ASSGN SYS012 A
DLBL OUTFILE A CMS TEST DATA (SYS012
FETCH TESTA (START
&IF &RETCODE = 100 &GOTO -RET100
&IF &RETCODE = 200 &GOTO -RET200
&EXIT &RETCODE
-RET100 &CONTINUE
.
.
.
-RET200 &CONTINUE
.
.
.
```

The &CONTROL and &ERROR control statements in the EXEC procedure ensure that if an error occurs during any part of the EXEC, the remainder of the EXEC does not execute, and the execution summary of the EXEC indicates the command that caused the error.

Note that for the DLBL command entered with the DSN ? operand, you must stack the response before issuing the DLBL command. In this example, since the DOS file-id has more than eight characters, you must use the &BEGSTACK control statement to stack it. If you use the &STACK control statement, the EXEC processor truncates all words to eight characters.

When your program is finished executing, the EXEC special variable &RETCODE indicates the contents of general register 15 at the time your program exited. You can use this value to perform additional steps in your EXEC procedure. Additional steps are indicated in the preceding example by ellipses.

For information on CMS EXEC procedures, see Appendix B, "The CMS EXEC Processor."

Chapter 11. Using Access Method Services and VSAM Under CMS and CMS/DOS

This section describes how you can use CMS to create and manipulate VSAM catalogs, data spaces, and files on OS and DOS disks using access method services. The CMS support is based on VSE and VSE/VSAM; this means that if you are an OS VSAM user and plan to use CMS to manipulate VSAM files, you are allowed to use those functions of access method services that are available under the access method services portion of VSE/VSAM. The control statements you can use are described in the publication *Using VSE/VSAM Commands and Macros.*

You can use CMS to:

- Execute the access method services utility programs for VSAM and SAM data sets on OS and DOS disks and minidisks. CMS can both read and write VSAM files using access method services.
- Compile and execute programs that read and write VSAM files from VSE programs
- Compile and execute programs that read and write VSAM files from OS programs.

VSAM files written under CMS are written using VSE/VSAM. Certain files written under CMS cannot be used directly by OS/VS VSAM. For information relative to compatibility between VSE/VSAM and OS/VS VSAM files, you should refer to the *VSE/VSAM General Information Manual.* None of the CMS commands normally used to manipulate CMS files are applicable to VSAM files, however. This includes such commands as PRINT, TYPE, EDIT, COPYFILE, and so on.

This section provides information on using the CMS AMSERV command with which you can execute access method services. Information is provided on using VSAM macros in CMS. The discussion is divided as follows:

- "Using the AMSERV command" contains general information.
- "Manipulating OS and DOS Disks for Use With AMSERV" describes how to use CMS commands with OS and DOS disks.
- "Defining DOS Input and Output Files" is for CMS/DOS users only.
- "Defining OS Input and Output Files" is for OS users only.
- "Using AMSERV Under CMS" includes notes and examples showing how to perform various access method services functions in CMS.
- "VSE/VSAM Macros" describes the macros and their support in CMS.
- "OS/VSAM Macros" describes the OSVSAM MACLIB supplied with CMS.

Executing VSAM Programs Under CMS

The commands that are used to define input and output data sets for access method services (DLBL) and for CMS/DOS users (ASSGN) are also used to identify VSAM input and output files for program execution. Information on executing

programs under CMS that manipulate VSAM files is contained in the program product documentation for the language processors. These publications are listed in the *VM/SP Introduction*.

Restrictions on the use of access method services and VSAM under CMS for OS and DOS users are listed in *VM/SP CMS Command and Macro Reference*, which also contains complete CMS and CMS/DOS command formats, operand descriptions, and responses for each of the commands described here.

When you are going to execute VSAM programs in CMS or CMS/DOS, you should remember to issue the DLBL command to identify the master catalog, as well as any other program input or output file you need to define.

Since VSE/VSAM Release 2, VSE/VSAM has reduced its dependency on explicit ASSGN, EXTENT, and DLBL information. In many cases, you no longer need to specify this information. Identification of the master catalog within CMS, however, still requires ASSGN and DLBL commands.

For complete information concerning the ASSGN, DLBL, and EXTENT requirements, refer to the *VSE/VSAM Programmer's Reference*.

In the discussion that follows, ASSGN, DLBL, and EXTENT information is included even though it may not be required.

Opening an ACB with a MACRF=ADR and subsequently issuing a GET or a PUT with KEYED ACCESS specified in the RPL when SHAREOPTION (4) is specified, is not allowed in VSE/VSAM Release 2. Likewise, opening an ACB with KEYED ACCESS and subsequently issuing a GET or a PUT with MACRF=ADR specified in the RPL when SHAREOPTION (4) is specified is not allowed. Please refer to *Using VSE/VSAM Commands and Macros* for more information.

VSE/VSAM supports the functions that were previously supported as well as the following enhancements:

- Volume ownership is enhanced so that multiple catalogs may own space in the same DASD volume if only one recoverable catalog owns space on the volume and only if one catalog resides on the volume.
- You can verify the syntax of the AMS commands without actually executing them by using the SYNCHK parameter of the AMS PARM command.
- Using the IGNOREERROR parameter of the AMS DELETE command, you can delete incomplete catalog information that may have resulted from a system failure during DEFINE or DELETE processing. When you specify the IGNOREERROR parameter of the AMS DELETE command, the PRINT option must be used on the CMS AMSERV command to send the listing to the virtual printer.
- A CMS VSAM user (with or without DOS set ON) may invoke the VSE/VSAM Catalog Check Service Aid to verify a complete catalog structure by invoking the CMS CATCHECK command.

Using the AMSERV Command

In CMS, you execute access method services utility programs with the AMSERV command, which has the basic format:

```
amserv filename
```

"filename" is the name of a CMS file that contains the control statements for access method services.

> **Note:** Throughout the remainder of this section the term "AMSERV" is used to refer to both the CMS AMSERV command and the OS/VS or VSE/VSAM access method services, except where a distinction is being made between CMS and access method services.

You create an AMSERV file with the CMS editor using a filetype of AMSERV and any filename you want; for example:

```
xedit mastcat amserv
```

The editor recognizes the filetype of AMSERV and so automatically sets the zone for your input lines at columns 2 and 72. The sample AMSERV file being created in the example above, MASTCAT AMSERV, might contain the following control statements:

```
DEFINE MASTERCATALOG (NAME (MYCAT) -
   VOLUME (123456) CYL(2) -
   FILE (IJSYSCT) )
```

> **Note:** The syntax of the control statements must conform to the rules for access method services, including continuation characters and parentheses. The only difference is that the AMSERV file does not contain a "/*" for a termination indicator.

Before you can execute the DEFINE control statement in this AMSERV example, you must define the output file, using the ddname IJSYSCT. You can do this using the DLBL command, if required by VSE/VSAM. Since the exact form required in the DLBL command varies according to whether you are an OS or a DOS user, separate discussions of the DLBL command are provided later in this section. All of the following examples assume that any disk data set or file that you are referencing with an AMSERV command will have been defined by a DLBL command, if required by VSE/VSAM.

When you execute the AMSERV command, the AMSERV control statement file can be on any accessed CMS disk; you do not need to specify the filemode and, if you are a DOS user, you do not need to assign SYSIPT. The task of locating the file and passing it to access method services is performed by CMS.

AMSERV Output Listings

When the AMSERV command is finished processing, you receive the CMS ready message, and if there was an error, the return code (from register 15) is displayed following the "R". For example:

```
R(00008);
```

or, if you are receiving the long form of the ready message, it appears:

```
R(00008); T=0.01/0.11 10:50:23
```

If you receive a ready message with an error return code, you should examine the output listing from AMSERV to determine the cause of the error.

AMSERV output listings are written in CMS files with a filetype of LISTING; by default, the filename is the same as that of the input AMSERV file. For example, if you have executed:

```
amserv mastcat
```

and the CMS ready message indicates an error return code, you should examine the file MASTCAT LISTING:

```
xedit mastcat listing
locate /idc/#=
```

Issuing the LOCATE subcommand twice to find the character string IDC will position you in the LISTING file at the first access method services message.

The publication *VSE/VSAM Messages and Codes* lists and explains all of the messages generated by access method services together with the associated return and reason codes.

Instead of editing the file, you could also use the TYPE command to display the contents of the entire file, so that you would be able to examine the input control statements as well as any error messages:

```
type mastcat listing
```

If you need to make changes to control statements before executing the AMSERV command again, use the CMS editor to modify the AMSERV input file.

If you execute the same AMSERV file a number of times, each execution results in a new LISTING file, which replaces any previous listing file with the same filename.

Output from PRINT, LISTCAT, and LISTCRA

When you use AMSERV to print a VSAM file, or to list catalog or recovery area contents using the PRINT, LISTCAT, or LISTCRA control statements, the output is written in a listing file on a CMS read/write disk, and not spooled to the printer unless you use the PRINT option of the AMSERV command:

```
amserv listcat (print
```

If you want to save the output, you should issue the AMSERV command without the PRINT option and then use the CMS PRINT command to print the LISTING file.

Controlling AMSERV Command Listings

The final disposition of the listing, as a printer or disk file, depends on how you enter the AMSERV command. If you enter the AMSERV command with no options, you get a CMS file with a filetype of LISTING and a filename equal to

that of the AMSERV input file. This LISTING file is usually written on your A-disk, but if your A-disk is full or not accessed, it is written on any other read/write CMS disk you have accessed.

If there is not enough room on your A-disk or any other disk, the AMSERV command issues an error message saying that it cannot write the LISTING file. If this happens, the LISTING file created may be incomplete and you may not be able to tell whether or not access method services actually completed successfully. In this case, after you have cleared some space on a read/write disk, you may have to execute an AMSERV PRINT or LISTCAT function to verify the completion of the prior job.

LISTING files take up considerable disk space, so you should erase them as soon as you no longer need them.

AMSERV Command Listing Options

If you do not want AMSERV to create a disk file from the listing, you can execute the AMSERV command with the PRINT option:

```
amserv myfile (print
```

The listing is spooled to your virtual printer, and no disk file is created. You might want to use this option if you are executing a PRINT or LISTCAT control statement and expect a very large output listing that you know cannot be contained on any of your disks.

You can also control the filename of the output listing file by specifying a second name on the AMSERV command line:

```
amserv listcat listcat1
```

In this example, the input file is LISTCAT AMSERV and the output listing is placed in a file named LISTCAT1 LISTING. A subsequent execution of this same AMSERV file:

```
amserv listcat listcat2
```

creates a second listing file, LISTCAT2 LISTING, so that the listing created from the first execution is not erased.

Manipulating OS and DOS Disks for Use with AMSERV

To use CMS VSAM and AMSERV, you can have OS or DOS disks in your virtual machine configuration. They can be assigned in your directory entry, or you can link to them using the CP LINK command. You must have read/write access to them in order to execute any AMSERV function or VSAM program that requires opening the file for output or update.

Before you can use an OS or DOS disk you must access it with the CMS ACCESS command:

```
access 200 d
```

The response from the ACCESS command indicates that the disk is in OS or DOS format:

```
D(200) R/W - OS
   -- or --
D(200) R/W - DOS
```

You can write on these disks only through AMSERV or through the execution of a program writing VSAM data sets. Once an OS disk is used with AMSERV or VSAM, CMS considers it a DOS disk, so regardless of whether you are an OS user, when you access or request information about a VSAM disk, CMS indicates that it is a DOS disk. You can still use the disk in an OS or DOS system for VSAM data set processing. Although the format is not changed, the disk is still subject to any incompatibilities that can currently exist between OS and DOS disks.

Data and Master Catalog Sharing

There are two meanings of "sharing" that must be defined clearly with respect to the CMS support of VSAM. The first is that of the SHAREOPTION parameter found in the DEFINE (and ALTER) command for access method services.

The SHAREOPTION keyword enables the VSAM user to define how a component will be shared within or across VSE partitions and VSE systems. Since CMS supports only a single partition environment, cross partition sharing has no meaning in the CMS environment. In addition, since CMS does not provide DASD sharing support, cross system sharing is not supported. Consequently, the SHAREOPTION parameter only has meaning within a CMS virtual machine (functional equivalent of a VSE partition).

The area of sharing most familiar to CMS users is that of disk (minidisk) read-sharing provided by CP. For the VSAM user under CMS, it is still possible to share disks in read-only mode in order to read-share VSAM components. However, there is a restriction with respect to the VSAM master catalog. That is, only one virtual machine may have the disk containing the master catalog in write status. This is necessary even if only read functions are being performed during the session. This is due to the master catalog updating read statistics at close time and, when necessary, writing a new control record in the catalog at open time.

Under CMS, it is possible to have the master catalog disk read-only. A programming modification (a bit in the ACB) was made to the DOS/VS VSAM code so that VSAM knows it is running under CMS. If this bit is on, VSAM will not write to the master catalog for either of the two cases described above. This allows one or more CMS virtual machines to share the VSAM master catalog. This assumes either no other virtual machine has the master catalog disk in write status or only one virtual machine (DOS, OS, or CMS) has it.

Multiple CMS users may have the VSAM master catalog disk in read-only status but only one virtual machine may have the same in write status. With respect to dataset sharing, there is only read-sharing for the CMS user.

Disk Compatibility

Since the CMS VSAM support writes VSAM datasets to DOS disks, the question of disk compatibility is not one between CMS and DOS nor between CMS or OS but rather between DOS and OS disks. In other words, because CMS actually uses VSE/VSAM for processing VSAM datasets, all disks used by CMS VSAM are DOS disks. For this reason, we need only discuss how DOS and OS disks are compatible and, because they are compatible, we can conclude that CMS and OS are also compatible.

In the format-4 DSCB, there is a bit in the VTOC Indicators (byte 59, bit 0) defined by OS/VS to indicate (when OFF) that a format-5 label is included in the VTOC. This bit is always ON under VSE because DOS does not maintain the format-5 label. This technique allows OS/VS to realize when the format-5 is invalid and that it must recompute free space and rewrite the format-5 so that device integrity is maintained.

Thus, if a disk originally was used (allocated) under OS/VS and, subsequently, with VSE further allocation could occur under VSE but with the format-5 ignored and, therefore, no longer valid. If the disk was then used under OS/VS and still further allocation performed, OS/VS would recognize the fact that the format-5 was not valid (contamination bit turned ON by VSE and would rewrite the format-5, turning the bit OFF.

In terms of space allocation, this shows that DOS and OS disks are compatible in that they are portable between the two systems, but one of the systems (OS/VS) must perform some extra processing (rewriting format-5) prior to using the disk if it intends to reallocate using the format-5.

DOS and OS disks containing VSAM datasets are no exception to this. OS and DOS disks containing VSAM datasets that are used (allocated) under CMS are portable among all three systems. Since CMS uses VSE/VSAM code, all disks used under CMS to process VSAM datasets become DOS disks in that the contamination bit is turned ON as it is when using VSE.

The term "minidisk" may be interchanged with the word "disk" in the above explanation if we are dealing with "virtual" VSE and OS/VS systems. However, real systems are not aware of, and do not support, minidisks.

VSE/VSAM uses physical record sizes ranging from .5K bytes to 8K bytes. All multiples of .5K bytes between those two values are supported. OS/VS VSAM, however, only supports physical record sizes of .5K, 1K, 2K, and 4K. Therefore, certain VSAM files written under CMS cannot be used directly by OS/VS VSAM.

It is necessary to distinguish between two types of allocation under VSAM.

1. The actual space allocation on the disk
2. Allocation within the dataset itself.

Space for VSAM components must be allocated on the DASD using the DEFINE commands. The only component for which the user is able to allocate space is for the master catalog, a user catalog, a data space, and a UNIQUE cluster. In defining the actual DASD space for components, there are parameters for the DEFINE SPACE command which allows the user to include a "secondary allocation" specification. These parameters (CYLINDERS, RECORDS, BLOCKS, TRACKS) have this secondary facility only as a syntactic compatibility with the OS/VS access method services commands. That is, VSE (and, therefore, CMS) does not perform secondary space allocation on a DASD.

The facility does exist under VSE (and CMS) to extend data or index components through already allocated data space, catalog extents, or UNIQUE cluster extents. Thus, the CYLINDERS, TRACKS, RECORDS, and BLOCKS parameters of the DEFINE commands for alternate indexes, clusters, and catalogs do not dynamically allocate DASD space but only extend a component through existing space.

Using VM/SP Minidisks

If you have a VM/SP minidisk in your virtual machine configuration, you can use it to contain VSAM files. Before you can use it, it must be formatted with the Device Support Facility program. When you request that a disk be added to your virtual machine configuration for use with VSAM files under CMS, you should indicate that it be formatted for use with OS or DOS. Or you can format it yourself using the Device Support Facility. How to do this is described under "Using Temporary Disks."

> **Note:** If you are an OS user, you should be careful about allocating space for VSAM on minidisks. Once you have used CMS AMSERV to allocate VSAM data space on a minidisk, you should not attempt to allocate additional space on that minidisk using an OS/VS system. OS does not recognize minidisks, and would attempt to format the entire disk pack and thus erase any data on it. To allocate additional space for VSAM, you should use CMS again.

Minidisk space allocation is fully described in the *VM/SP Planning Guide and Reference.*

Using The LISTDS Command

For OS or DOS disks or minidisks, you can use the LISTDS command to determine the extents of free space available for use by VSAM. You can also determine what space is already in use. You can use this information to supply the extent information when you define VSAM files.

The options used with VSAM disks are:

- EXTENT, to find out what extents are in use, and
- FREE, to find out what extents are available.

For example, if you have an OS disk accessed as a G-disk, and you enter:

```
listds g (extent
```

The response might look like:

```
EXTENT INFORMATION FOR 'VTOC' ON 'G' DISK:
SEQ TYPE  CYL-HD(RELTRK) TO CYL-HD(RELTRK)   TRACKS
000 VTOC  099 00  1881     099 18   1899        19

EXTENT INFORMATION FOR 'PRIVAT.CORE.IMAGE.LIB' ON 'G' DISK:
SEQ TYPE  CYL-HD(RELTRK) TO CYL-HD(RELTRK)   TRACKS
000 DATA  000 01      1     049 18    949       949

EXTENT INFORMATION FOR 'SYSTEM.WORK.FILE.NO.6' ON 'G' DISK:
SEQ TYPE  CYL-HD(RELTRK) TO CYL-HD(RELTRK)   TRACKS
000 DATA  050 00    950     051 18    987        38
```

You could also determine the extent for a particular data set:

```
listds ? * (extent

DMSLDS220R ENTER DATA SET NAME:

system.recorder.file
```

The response might look like:

```
EXTENT INFORMATION FOR 'SYSTEM RECORDER FILE' ON 'F' DISK:
SEQ TYPE  CYL-HD(RELTRK) TO CYL-HD(RELTRK)   TRACKS
000 DATA  102 00   1938     102 18   1956       19
002 DATA  010 06    206     010 08    208        3
```

LISTDS searches all minidisks accessed until it locates the specified data set. In this example, the data set occupies two separate extents on disk F. If the data set is a multivolume data set, extents on all accessed volumes are located and displayed.

If you want to find the free extents on a particular disk, enter:

```
listds g (free
```

The response might look like:

```
FREESPACE EXTENTS FOR 'G' DISK:
CYL-HD(RELTRK) TO CYL-HD(RELTRK)   TRACKS
052 00    988     052 01    989         2
054 02   1028     080 00   1520       493
081 01   1540     098 18   1880       341
```

You can use this information when you allocate space for VSAM files. If you enter:

```
listds * (free
```

CMS lists all the free space available on all of your accessed disks.

Using Temporary Disks

When you need extra space on a temporary basis for use with CMS VSAM and AMSERV, you can use the CP DEFINE command to define a temporary minidisk and then use the Device Support Facilities program to format it. Refer to the *Device Support Facilities User's Guide and Reference.* Once formatted and accessed, it is available to your virtual machine for the duration of your terminal session or until you detach it using the CP DETACH command. Remember that anything placed on a temporary disk is lost, so that you should copy output that you want to keep onto permanent disks before you log off.

Formatting a Temporary Disk

The example below shows a control statement file and an EXEC procedure that, together, can be used to format a minidisk using the Device Support Facility. For a complete description of the control statements used, refer to the *Device Support Facilities User's Guide and Reference.*

The input control statements for the Device Support Facility should be placed in a CMS file, so that they can be punched to your virtual card reader. For this example, suppose the statements are in a CMS file named TEMP DSF:

```
 INIT UNIT(198) DEVTYP(3340) PRG NVFY VOLID(123456) DVTOC(9,7,5) -
   MIMIC (MINI(10))
```

Note:
The example above begins in column 2.

Now consider the CMS file named TEMPDISK EXEC:

```
&CONTROL OFF
&ERROR &EXIT 100
CP DEFINE T3340 198 10
CP CLOSE READER
CP PURGE READER CLASS I
CP SPOOL PUNCH TO * CLASS I CONT NOHOLD
PUNCH IPL DSF * (NOH)
PUNCH TEMP DSF * (NOH)
CP SPOOL PUNCH NOCONT CLOSE
CP SPOOL READER CLASS I NOHOLD
CP IPL 00C CLEAR ATTN
```

You execute this procedure by entering the filename of the EXEC:

```
tempdisk
```

When the final line of this EXEC is executed, the Device Support Facility is in control.

```
ICK005E DEFINE INPUT DEVICE, REPLY 'DDDD,CUU OR CONSOLE'
ENTER INPUT/COMMAND:
```

You should enter:

```
2540,00c
```

to indicate that the control statements should be read from your card reader, which is a virtual 2540 device at virtual address 00C.

```
ICK006E DEFINE OUTPUT DEVICE, REPLY 'DDDD,CUU OR CONSOLE'
ENTER INPUT/COMMAND:
```

You should enter:

```
console
```

to indicate that the utility output should sent to your console.

```
ICK003D REPLY U TO ALTER VOLUME 198 CONTENTS, ELSE T
ENTER INPUT/COMMAND:
```

Reply to ICK003D:

```
u
```

to continue the execution.

When the Device Support Facilities program is completed, your virtual machine is in a wait state and you must reload CMS (with the IPL command) to begin virtual machine execution. You can then access the temporary disk:

```
acc 198 c
```

and CMS responds:

```
C (198) R/W - DOS
```

Defining DOS Input and Output Files

Note: This information is for VSE/VSAM users. OS/VS VSAM users should refer to the section "Defining OS Input and Output Files." You may use the DLBL command to define VSAM input and output files for both the AMSERV command and for program execution. The operands required on the DLBL command are:

```
dlbl ddname filemode DSN datasetname (options SYSxxx
```

where "ddname" corresponds to the FILE parameter in the AMSERV file and "datasetname" corresponds to the entry name or filename of the VSAM file.

There are several options you can use when issuing the DLBL command to define VSAM input and output files. These are:

- VSAM, which you must use to indicate that the file is a VSAM file.

  Note: You do not have to use the VSAM option to identify a file as a VSAM file if you are using any of the other options listed here, since they imply that the file is a VSAM file. In addition, the ddnames (filenames) IJSYSCT and IJSYSUC also indicate that the file being defined is a VSAM file.

- EXTENT, which you may use when you are defining a catalog or a VSAM data space; you are prompted to enter the volume information. This option effectively provides the function of the EXTENT card in VSE.

- MULT, which you must use in order to access a multivolume VSAM file; you are prompted to enter the extent information.

- CAT, which you can use to identify a catalog which contains the entry for the VSAM file you are defining.

- BUFSP, which you can use to specify the size of the buffers VSAM should use during program execution.

Options are entered following the open parenthesis on the DLBL command line, with the SYSxxx:

```
assgn sys003 e
dlbl file1 b1 dsn workfile (extent cat cat2 sys003
```

Using VSAM Catalogs

While you are developing and testing your VSAM programs in CMS, you may find it convenient to create and use your own master catalog, which may be on a CMS minidisk. VSAM catalogs, like any other cluster, can be shared read-only among several users.

You name the VSAM master catalog for your terminal session using the logical unit SYSCAT in the ASSGN command and the ddname IJSYSCT for the DLBL command. For example, if your VSAM master catalog is located on a DOS disk you have accessed as a C-disk, you would enter:

```
assgn syscat c
dlbl ijsysct c dsn mastcat (syscat
```

Note: When you use the ddname IJSYSCT you do not need to specify the VSAM option on the DLBL command.

You must identify the master catalog at the start of every terminal session. If you are always using the same master catalog, you might include the ASSGN and DLBL commands in an EXEC procedure or in your PROFILE EXEC. You could also include the commands necessary to access the DOS system residence volume and enter the CMS/DOS environment:

```
ACCESS 350 Z
SET DOS ON Z (VSAM
ACCESS 555 C
ASSGN SYSCAT C
DLBL IJSYSCT C DSN MASTCAT (SYSCAT PERM
```

You should use the PERM option so that you do not have to reset the master catalog assignment after clearing previous DLBL definitions.

You must use the VSAM option on the SET DOS ON command line if you want to use any access method services function or access VSAM files.

Defining a Master Catalog

The sample ASSGN and DLBL commands used in the above EXEC are almost identical to those you issue to define a master catalog using AMSERV. The only difference is the EXTENT option which lists the data spaces that this master catalog is to control.

As an example, suppose that you have a 30-cylinder 3330 minidisk assigned to you to use for testing your VSAM programs under CMS. Assuming that the minidisk is in your directory at address 333, you should first access it:

```
access 333 d
D(333) R/W - OS
```

If you formatted the minidisk yourself, you know what its label is. If not, you can find out what the label is by using the CMS command:

```
query search
```

The response might be:

```
USR191   191  A    R/W
DOS333   333  D    R/W - OS
SYS190   190  S    R/O
SYS19E   19E  Y/S  R/O
```

Use the label DOS333 in the VOLUMES parameter in the MASTCAT AMSERV file:

```
DEFINE MASTERCATALOG -
   (NAME  (MASTCAT ) -
    VOLUME (DOS333) -
    CYL (4) -
    FILE (IJSYSCT)    )
```

To find out what extents on the minidisk you can allocate for VSAM, use the LISTDS command with the FREE option:

```
listds d (free
```

The response from LISTDS might look like this:

```
FREESPACE INFORMATION FOR 'D' DISK:
CYL-HD(RELTRK)  TO CYL-HD(RELTRK)    TRACKS
000 01       1     000 09       9         9
000 11      11     029 18     569       560
```

From this response, you can see that the volume table of contents (VTOC) is located on the first cylinder, so you can allocate cylinders 1 through 29 for VSAM:

```
assgn syscat d
dlbl ijsysct d dsn mastcat (syscat perm extent
DMSDLB331R ENTER EXTENT SPECIFICATIONS:
19 551
   (null line)
```

After entering the extents, in tracks, giving the relative track number of the first track to be allocated followed by the number of tracks, you must enter a null line to complete the command. A null line is required because, when you enter multiple extents, entries may be placed on more than one line. If you do not enter a null line, the next line you enter causes an error, and you must re-enter all of the extent information.

> **Note:** As in VSE the extents must be on cylinder boundaries, and you cannot allocate cylinder 0.

Now you can issue the AMSERV command:

```
amserv mastcat
```

A ready message with no return code indicates that the master catalog is defined. You do not need to reissue the ASSGN and DLBL commands in order to use the master catalog for additional AMSERV functions.

Defining User Catalogs

You can use the AMSERV command to define private catalogs and spaces for them, also. The procedures for determining what space you can allocate are the same as those outlined in the example of defining a master catalog.

For a user catalog, you may use any programmer logical unit, and any ddname:

```
access 199 e
listds e (free
.
.
.
assgn sys001 e
dlbl cat1 e dsn private.cat1 (sys001 extent perm
.
.
.
amserv usercat
```

The file USERCAT AMSERV might contain the following:

```
DEFINE USERCATALOG -
     (NAME (PRIVATE.CAT1) -
      FILE  (IJSYSUC) -
      CYL (4) -
      VOLUME (DOSVS2) ) -
      CATALOG (MASTCAT)
```

After this AMSERV command has completed successfully you can use the catalog PRIVATE.CAT1. When you issue a DLBL command to identify a cluster or data set cataloged in this catalog, you must identify the catalog using the CAT option on the DLBL command for the file:

```
assgn sys100 c
dlbl file2 c dsn ? (sys100 cat cat1
```

Or, you can define this catalog as a job catalog.

Using a Job Catalog

If you want to set up a user catalog as a job catalog so that it will be searched during all subsequent jobs, you can define the catalog using the special ddname IJSYSUC. For example:

```
assgn sys101 c
dlbl ijsysuc c dsn private.cat1 (sys101 perm
```

If you defined a user catalog (IJSYSUC) for a terminal session and you use the AMSERV command to access a VSAM file, the user catalog takes precedence over the master catalog. This means that for files that already exist, only the user catalog is searched. When you define a cluster, it is cataloged in the user catalog, rather than in the master catalog, unless you use the CAT option to override it.

If you want to use additional catalogs during a terminal session, you first define them just as you would any other VSAM file:

```
assgn sys010 f
dlbl mycat2 f dsn private.cat2 (sys010 vsam
```

Then, when you enter the DLBL command for the VSAM file that is cataloged in PRIVATE.CAT2 use the CAT option to refer to the ddname of the catalog:

```
assgn sys011 f
dlbl input f dsn input.file (sys011 cat mycat2
```

If you want to stop using a job catalog defined as IJSYSUC, you can clear it using the CLEAR option of the DLBL command:

```
dlbl ijsysuc clear
```

Then, the master catalog becomes the job catalog for files not defined with the CAT option.

Catalog Passwords

When you define passwords for VSAM catalogs in CMS, or when you use CMS to access VSAM catalogs that have passwords associated with them, you must supply the password from your terminal when the AMSERV command executes. The message that you receive to prompt you for the password is the same message you receive when you execute access method services:

```
4221A ATTEMPT 1 OF 2. ENTER PASSWORD FOR JOB AMSERV   FILE catalog
```

When you enter the proper password, AMSERV continues execution.

Verifying A Catalog Structure

As a CMS VSAM user (with or without DOS set ON), you can use the CMS CATCHECK command to invoke the VSE/VSAM Catalog Check Service Aid to verify a complete catalog structure. If you do not specify a catalog name with the CATCHECK command, the catalog specified with the DLBL command is used. CATCHECK produces a print file containing the catalog analysis. For example, issuing:

```
dlbl ijsysuc f dsn private.cat1 (vsam
```

and

```
catcheck
```

results in a print file containing the VSE/VSAM Catalog Check output.

If you had issued only a DLBL for the master catalog, issuing:

```
catcheck private.cat1
```

produces the same result.

Defining and Allocating Space for VSAM files

You can use CMS AMSERV to allocate additional data spaces for VSAM. To use the DEFINE SPACE control statement, you must have defined the catalog that is to control the space, and you must have the volume or volumes on which the space is to be allocated mounted and accessed.

For example, suppose you have a DOS-formatted 3330 disk attached to your virtual machine at virtual address 255. After accessing the disk and determining the free space on it, you could create a file named SPACE AMSERV:

```
DEFINE SPACE -
   (FILE (FILE1) -
    TRACKS (1900) -
    VOLUME (123456) ) -
    CATALOG (PRIVATE.CAT2 CAT2)
```

Before executing this AMSERV file, define PRIVATE.CAT2 as a user catalog using the ddname CAT2, and then define the ddname for the FILE parameter:

```
access 255 c
assgn sys010 c
dlbl cat2 c dsn private.cat2 (sys010 vsam
assgn sys011 c
dlbl file1 c (extent sys011 cat cat2
amserv space
```

You do not need to enter a data set name to define the space. When CMS prompts you for the extents of the space you can enter the extent specifications:

```
DMSDLB331R ENTER EXTENT SPECIFICATIONS:
190 1900
.
.
.
```

When you define space for VSAM, you should be sure that the VOLUMES parameter and the space allocation parameter (whether CYLINDER, TRACKS, BLOCKS, or RECORDS) in the AMSERV file agrees with the information you provide in the DLBL command. All data extents must begin and end on cylinder boundaries. Any additional space you provide in the extent information that is beyond what you specified in the AMSERV file is claimed by VSAM.

Specifying Multiple Extents

When you are specifying extents for a master catalog, data space, or unique file, you can specify up to 16 extents on a volume for a particular space. When prompted by CMS to enter the extents, you must separate different extents by commas or place them on different lines. To specify a range of extents in the above example, you can enter:

```
dlbl file1 c (extent sys011
190  190, 570  190, 1900 1520
   (null line)
  -- or --
dlbl file1 c (extent sys011
190  190
570  190
1900 1520
   (null line)
```

Again, the first number entered for each extent represents the relative track for the beginning of the extent and the second number indicates the number of tracks.

Specifying Multivolume Extents

You can define spaces that span up to nine volumes for VSAM files; all of the volumes must be accessed and assigned when you issue the DLBL command to define or identify the data space.

You should remember, though, that if you are using AMSERV and you do not use the PRINT option, you must have a read/write CMS disk so that AMSERV can write the output LISTING file.

If you are defining a new multivolume data space or unique cluster, you must specify the extents on each volume that the data is to occupy (starting track and number of tracks), followed by the disk mode letter at which the disk is accessed and the programmer logical unit to which the disk is assigned:

```
access 135 b
access 136 c
access 137 d
assgn  sys001 b
assgn  sys002 c
assgn  sys003 d
dlbl newfile b (extent sys001
DMSDLB331R ENTER EXTENT SPECIFICATIONS:
100 60 b sys001, 400 80 b sys001, 60 40 d sys003
2000 100 c sys002
   (null line)
```

If you specify more than one extent on the same line, the extents must be separated by commas; if you enter a comma at the end of a line, it is ignored. Different extents for the same volume must be entered consecutively.

> **Note:** In the preceding example, the extent information is for 2314 disks; these extents are also on cylinder boundaries.

When you enter multivolume extents you can use a default mode. For example:

```
dlbl newfile b (extent sys001
DMSDLB331R ENTER EXTENT SPECIFICATIONS:
100 60, 400 80, 60 40 d sys003,
2000 100 c sys002
   (null line)
```

Any extents you enter without specifying a mode letter and SYSxxx value default to the mode and SYSxxx on the DLBL command line, in this case, the B-disk, SYS001.

If you make any errors issuing the DLBL command or extent information, you must re-enter the entire command sequence.

Identifying Existing Multivolume Files: When you issue a DLBL command to identify an existing multivolume VSAM file, you must use the MULT option of the DLBL command:

```
dlbl old b1 dsn ? (sys002 mult
DMSDLB220R ENTER DATA SET NAME:
dostest.file
DMSDLB330R ENTER VOLUME SPECIFICATIONS:
c sys004, d sys003
e sys007
  (null line)
```

When you enter the DLBL command you should specify the mode letter and logical unit for the first volume on the command line. When you enter the MULT option you are prompted to enter additional specifications for the remaining extents. In the preceding example, the data set has extents on disks accessed as B-, C-, D-, and E-disks.

Using Tape Input and Output

If you are using AMSERV for a function that requires tape input and/or output, you must have the tape(s) attached to your virtual machine. The valid addresses for tapes are 181, 182, 183, and 184. When referring to tapes, you can also refer to them using their CMS symbolic names TAP1, TAP2, TAP3, and TAP4.

For AMSERV functions that use tape input/output, the TLBL control statement is simulated by building a dummy DLBL containing a user-supplied ddname (filename). CMS does not read tape labels and does not recognize tape data set names.

When you invoke the AMSERV command, you must use the TAPIN or TAPOUT option to specify the tape device being used:

```
amserv export (tapout 181
```

In this example, the output from the AMSERV control statements in a file named EXPORT goes to a tape at virtual address 181. CMS prompts you to enter the ddname:

```
DMSAMS367R ENTER TAPE OUTPUT DDNAMES:
```

After you enter the ddname specified on the FILE parameter in the AMSERV file and press the carriage return, the AMSERV command executes.

AMSERV opens all tape files as standard labelled tapes or non-labelled tapes. If you are using standard labelled tapes, you need to specify a LABELDEF command with AMSERV. The LABELDEF command is the CMS/DOS equivalent of VSE TLB control statement. The LABELDEF command is used to specify information in VOL1 and HDR1 labels on the tape. See the description of the LABELDEF command in Chapter 6, "Using Real Printers, Punches, Readers, and Tapes" for more information on this command.

You should use the same name for the filename on your LABELDEF command as you do for the ddname you enter in reply to message DMSAMS367R (the ddname specified on the FILE parameter in the AMSERV file). However, the LABELDEF command must be issued before the AMSERV command. The following sequence of commands might be used when you have standard labelled tape output.

```
assgn sys005 tap1
tape rew (181
assgn syscat e
assgn sys006 e
labeldef catout fid catfile volid amserv
dlbl ijsysct e dsn mastcat (syscat vsam
dlbl catin e dsn file (sys006 vsam
amserv repro (tapout 181

DMSAMS367R ENTER TAPE OUTPUT DDNAMES

catout
```

Note: If you do not care what is written in a tape output label or do not want input labels checked, you can specify a LABELDEF with no parameters other than filename. When you enter:

```
labeldef intape
```

for an input tape with ddname INTAPE , the standard labels on the tape to be skipped without any checking. A similar statement for output writes tape labels with default values (see the description of the LABELDEF command in Chapter 6, "Using Real Printers, Punches, Readers, and Tapes" on page 6-1.)

If you use non-labelled tapes, LABELDEF is not required.

Reading VSAM Tape Files

When you create a tape in CMS using AMSERV, CMS writes a tape mark preceding each output file that it writes. When the same tape is read using AMSERV under CMS, HDR1 and VOL1 labels are checked using the LABELDEF command you provide. If you read this tape in a real VSE system, you should use a TLBL card instead of the LABELDEF command.

Similarly, when you create a tape under a VSE system using access method services, if the tape is created with standard labels, CMS AMSERV has no difficulty reading it.

The only time you should worry about positioning a tape created by AMSERV is when you want to read the tape using a method other than AMSERV, for example, the MOVEFILE command. Then, you must forward space the tape past the label, using the CMS TAPE command before you can read it.

Defining OS Input and Output Files

Note: This information is for OS/VS VSAM users only. VSE/VSAM users should refer to "Defining DOS Input and Output Files" for information on defining files for use with VSAM.

The OS/VS VSAM user should bear in mind that CMS uses VSE/VSAM to manipulate VSAM files. The VSAM and AMS statements that can be used are described in the publication *Using VSE/VSAM Commands and Macros.*

In addition, there are certain incompatibilities between VSE/VSAM and OS/VS VSAM. For a description of these incompatibilities, refer to the *VSE/VSAM General Information Manual.*

If you are going to use access method services to manipulate VSAM or SAM files or you are going to execute VSAM programs under CMS, use the DLBL command to define the input and output files. The basic format of the DLBL command is:

```
DLBL ddname filemode DSN datasetname (options
```

where ddname corresponds to the FILE parameter in the AMSERV file and datasetname corresponds to the entry name of the VSAM file, that is, the name specified in the NAME parameter of an access method services control statement.

If you are using a CMS file for AMSERV input or output, use the CMS operand and enter CMS file identifiers as follows:

```
dlbl mine a cms out file1 (vsam
```

The maximum length allowed for ddnames under CMS VSAM is seven characters. This means that if you have assigned eight-character ddnames (or filenames) to files in your programs, only the first seven characters of each ddname are used. So, if a program refers to the ddname OUTPUTDD, you should issue the DLBL command for a ddname of OUTPUTD. Since you can encounter problems with a program that contains ddnames with the same first seven characters, you should recompile those programs using seven-character ddnames.

There are several options you can use when issuing the DLBL command to define VSAM input and output files. These are:

- VSAM, which you must use to indicate that the file is a VSAM file.

  Note: You do not have to use the VSAM option to identify a file as a VSAM file if you are using any of the other options listed here, since they imply that the file is a VSAM file. In addition, the ddnames (filenames) IJSYSCT and IJSYSUC also indicate that the file being defined is a VSAM file.

- EXTENT, which you can use when you are defining a catalog or a VSAM data space; you are prompted to enter the volume information.

- MULT, which you must use in order to access a multivolume VSAM file; you are prompted to enter the extent information.

- CAT, which you can use to identify a catalog which contains the entry for the VSAM file you are defining.

- BUFSP, which you can use to specify the size of the buffers VSAM should use during program execution.

Allocating Extents on OS Disks and Minidisks

When you use access method services to manipulate VSAM files under OS, you do not have to worry about allocating the real cylinders and tracks to contain the files. You can, however, use CMS commands to indicate which cylinders and tracks should contain particular VSAM spaces when you use the DEFINE control statement to define space.

Extents for VSAM data spaces can be defined, in AMSERV files, in terms of cylinders, tracks, or records. Extent information you supply to CMS when executing AMSERV must always be in terms of tracks. When you define data spaces or unique clusters, the extent information (number of cylinders, tracks, or records) in the AMSERV file must match the extents you supply when you issue the DLBL command to define the file. When you supply extent information for the master catalog, any extents you enter in excess of those required for the catalog are claimed by the catalog and used as data space.

CMS does not make secondary space allocation for VSAM data spaces. If you execute an AMSERV file that specifies a secondary space allocation, CMS ignores the parameter.

When you use the DLBL command to define VSAM data space, you can use the EXTENT option, which indicates to CMS that you are going to enter data extents. For example, if you enter:

```
dlbl space b (extent
```

CMS prompts you to enter the extents:

```
DMSDLB331R ENTER EXTENT SPECIFICATIONS:
```

When you enter the extents, you specify the relative track number of the first track of the extent, followed by the number of tracks. For example, if you are allocating an entire 2314 disk, you would enter:

```
20 3980
   (null line)
```

You can never write on cylinder 0, track 0; and, since VSAM data spaces must be allocated on cylinder boundaries, you should never allocate cylinder 0. Cylinder 0 is often used for the volume table of contents (VTOC) as well, so it is always best to begin defining space with cylinder 1.

You can determine which disk extents on an OS disk or minidisk are available for allocation by using the LISTDS command with the FREE option, which also indicates the relative track numbers, as well as actual cylinder and head numbers.

Using VSAM Catalogs

While you are developing and testing your VSAM programs in CMS, you may find it convenient to create and use your own master catalog, which may be on a CMS minidisk. VSAM catalogs, like any other cluster, can be shared read-only among several users.

You name the VSAM master catalog for your terminal session using the ddname IJSYSCT for the DLBL command. For example, if your VSAM master catalog is located on an OS disk you have accessed as a C-disk, you would enter:

```
dlbl ijsysct c dsn master catalog (perm
```

You must define the master catalog at the start of every terminal session. If you are always using the same master catalog, you might include the DLBL command you need to define it in your PROFILE EXEC:

```
ACCESS 555 C
DLBL IJSYSCT C DSN MASTCAT (PERM
```

You should use the PERM option so that you do not have to reset the master catalog assignment after clearing previous DLBL definitions. The command:

```
dlbl * clear
```

clears all file definitions except those entered with the PERM option.

Defining a Master Catalog

The sample DLBL command used in the preceding example is almost identical with the one you would issue to define a master catalog using AMSERV. The only difference is that you can enter the EXTENT option so that you can list the data spaces that this master catalog is to control.

As an example, suppose that you have a 30-cylinder 3330 minidisk assigned to you to use for testing your VSAM programs under CMS. Assuming that the minidisk is in your directory at address 333, you should first access it:

```
access 333 d
D(333) R/W - OS
```

If you formatted the minidisk yourself, you know what label you assigned it; if not, you can find out the label assigned to the disk by issuing the CMS command:

```
query search
```

The response might be:

```
USR191  191  A    R/W
VSAM03  333  D    R/W - OS
SYS109  190  S    R/O
SYS19E  19E  Y/S  R/O
```

Use the volume label VSAM03 in the MASTCAT AMSERV file:

```
DEFINE MASTERCATALOG -
   (NAME   (MASTCAT ) -
    VOLUME (VSAM03) -
    CYL (4) -
    FILE (IJSYSCT)    )
```

To find out what extents on this minidisk you can allocate for VSAM, use the LISTDS command with the FREE option:

```
listds d (free
```

The response from LISTDS might look like this:

```
FREESPACE INFORMATION FOR 'D' DISK:
CYL-HD(RELTRK) TO CYL-HD(RELTRK)   TRACKS
000 01      1     000 09      9        9
000 11     11     029 18    569      560
```

From this response, you can see that the VTOC is located on the first cylinder, so you can allocate cylinders 1 through 29 for VSAM:

```
dlbl ijsysct d dsn mastcat (perm extent
DMSDLB331R ENTER EXTENT SPECIFICATIONS:
19 551
    (null line)
```

After entering the extents, in tracks, giving the relative track number of the first track to be allocated followed by the number of tracks, you must enter a null line to complete the command. (A null line is required because, when you enter multiple extents, entries may be placed on more than one line.)

Now you can issue the AMSERV command:

```
amserv mastcat
```

A ready message with no return code indicates that the master catalog is defined. You do not need to reissue the DLBL command in order to identify the master catalog for additional AMSERV functions.

Defining User Catalogs

You can use the AMSERV command to define private catalogs and spaces for them. The procedures for determining what space you can allocate are the same as those outlined in the example of defining a master catalog.

To define a user catalog, you can assign any ddname you want:

```
access 199 e
listds e (free
.
.
.
dlbl cat1 e dsn private.cat1 (extent
.
.
.
amserv usercat
```

The file USERCAT AMSERV might contain the following:

```
DEFINE USERCATALOG -
     (NAME (PRIVATE.CAT1) -
      FILE  (CAT1) -
      CYL (4) -
      VOLUME (OSVSAM) ) -
      CATALOG (MASTCAT)
```

dfter this AMSERV command has completed successfully you can use the catalog PRIVATE.CAT1. When you define a file cataloged in it, you identify it using the CAT option on the DLBL command:

```
dlbl file2 e dsn ? (cat cat1
```

Or, you can define it as a job catalog.

Using a Job Catalog

During a terminal session, you may be referencing the same private catalog many times. If this is the case, you can identify a job catalog by using the ddname IJSYSUC. Then, that catalog is searched during all subsequent jobs, unless you override it using the CAT option when you use the DLBL command to define a file.

If you defined a user catalog (IJSYSUC) for a terminal session and you use the AMSERV command to access a VSAM file, the user catalog takes precedence over the master catalog. This means that for files that already exist, the job catalog is searched. When you define a cluster, it is cataloged in the job catalog, rather than in the master catalog, unless you use the CAT option to override it. CMS never searches more than one VSAM catalog.

You should use the CAT option to name a catalog when the AMSERV file you are executing references, with the CATALOG parameter, a catalog that is not defined either as the master catalog or as a user catalog.

If you want to use additional catalogs during a terminal session, you first define them just as you would any other VSAM file:

```
dlbl mycat2 f dsn private.cat2 (vsam
```

Then, when you enter the DLBL command for the VSAM file that is cataloged in PRIVATE.CAT2 use the CAT option to refer to the ddname of the catalog:

```
dlbl input f dsn input.file (cat mycat2
```

If you want to stop using a job catalog defined with the ddname IJSYSUC, you can clear it using the CLEAR option of the DLBL command:

```
dlbl ijsysuc clear
```

or, you can assign the ddname IJSYSUC to some other catalog. If you clear the ddname for IJSYSUC, then the master catalog becomes the job catalog.

Catalog Passwords

When you define passwords for VSAM catalogs in CMS, or when you use CMS to access VSAM catalogs that have passwords associated with them, you must supply the password from your terminal when the AMSERV command executes. The message that you receive to prompt you for the password is the same message you receive when you execute access method services:

```
4221A ATTEMPT 1 OF 2. ENTER PASSWORD FOR JOB AMSERV   FILE catalog
```

When you enter the proper password, AMSERV continues execution.

Verifying a Catalog Structure

As a CMS VSAM user (with or without DOS set ON), you can use the CMS CATCHECK command to invoke the VSE/VSAM Catalog Check Service Aid to verify a complete catalog structure. If you do not specify a catalog name with the CATCHECK command, the catalog specified with the DLBL command is used. CATCHECK produces a print file containing the catalog analysis. For example, issuing:

```
dlbl ijsysuc f dsn private.cat1 (vsam
```

and

```
catcheck
```

results in a print file containing the VSE/VSAM Catalog Check output.

If you had issued only a DLBL for the master catalog, issuing:

```
catcheck private.cat1
```

produces the same result.

Defining and Allocating Space for VSAM files

You can use CMS AMSERV to allocate additional data spaces for VSAM. To use the DEFINE SPACE control statement, you must have defined either the master catalog or a user catalog which will control the space, and you must have the volume or volumes on which the space is to be allocated mounted and accessed.

For example, suppose you have an OS 3330 disk attached to your virtual machine at virtual address 255. After accessing the disk and determining the free space on it, you could create a file named SPACE AMSERV:

```
DEFINE SPACE -
   (FILE (FILE1) -
    TRACKS (1900) -
    VOLUME (123456) ) -
    CATALOG (PRIVATE.CAT2 CAT2)
```

To execute this AMSERV file, you must define PRIVATE.CAT2 using the ddname CAT2, and then define the ddname for the file:

```
access 255 c
dlbl cat2 c dsn private.cat2 (vsam
dlbl file1 c (extent cat cat2
```

You do not need to enter a data set name to define the space. When CMS prompts you for the extents of the space, you can enter the extent specifications:

```
DMSDLB331R ENTER EXTENT SPECIFICATIONS:
190 1900
.
.
.
```

When you define space for VSAM, you should be sure that the VOLUMES parameter and the space allocation parameter (whether CYLINDER, TRACKS, BLOCKS, or RECORDS) in the AMSERV file agree with the track information you provide in the DLBL command.

Specifying Multiple Extents

When you are specifying extents for a master catalog, data space, or unique file, you can specify up to 16 extents on a volume for a particular space. When prompted by CMS for the extents, you must separate the different extents by commas, or place them on different lines. To specify a range of extents in the above example, you could enter:

```
dlbl file1 c (extent
190  190, 570  190, 1900 1520
   (null line)
  -- or --
dlbl file1 c (extent
190  190
570  190
1900 1520
   (null line)
```

Again, the first number entered for each extent represents the relative track for the beginning of the extent and the second number indicates the number of tracks.

Specifying Multivolume Extents

You can define spaces that span up to nine volumes for VSAM files; all of the volumes must be accessed and assigned when you issue the DLBL command to define or identify the data space.

You should remember, though, that if you are using AMSERV and you do not use the PRINT option, you must have a read/write CMS disk so that AMSERV can write the output LISTING file.

If you are defining a new multivolume data space or unique cluster, you must specify the extents on each volume that the data is to occupy (starting track and number of tracks), followed by the disk mode letter at which the disk is assigned:

```
access 135 b
access 136 c
access 137 d
dlbl newfile b (extent
DMSDLB331R ENTER EXTENT SPECIFICATIONS:
100 60 b, 400 80 b, 60 40 d,
2000 100 c
   (null line)
```

If you enter more than one extent on the same line, the extents must be separated by commas; if you enter a comma at the end of a line, it is ignored. Different extents for the same volume must be entered consecutively.

> **Note:** In this example, the extent information is for 2314 disks and that these extents are also on cylinder boundaries.

When you enter multivolume extents, you do not have to enter a mode letter for those extents on the disk identified in the DLBL command. For the extents on disk B in the above example, you could enter:

```
dlbl newfile b (extent
DMSDLB331R ENTER EXTENT SPECIFICATIONS:
100 400 80, 60, 60 40 d
2000 100 c
   (null line)
```

If you make any errors issuing the DLBL command or extent information, you must reissue the entire command sequence.

Identifying Existing Multivolume Files:

When you issue a DLBL command to identify an existing multivolume VSAM file, you must use the MULT option of the DLBL command sequence:

```
dlbl old b1 dsn ? (mult
DMSDLB220R ENTER DATASET NAME:
vsamtest.file
DMSDLB330R ENTER VOLUME SPECIFICATIONS:
c, d
e
   (null line)
```

When you enter the DLBL command you should specify the mode letter for the first disk volume on the command line. When you enter the MULT option you are prompted to enter additional specifications for the remaining extents. In the above example, the data set has extents on disks accessed as B-, C-, D-, and E-disks.

Using Tape Input and Output

If you are using AMSERV for a function that requires tape input and/or output, you must have the tape(s) attached to your virtual machine. The valid addresses for tapes are 181, 182, 183, and 184. When referring to tapes, you can also refer to them using their CMS symbolic names TAP1, TAP2, TAP3, and TAP4.

When you use AMSERV to create or read a tape, you supply the ddname for the tape device interactively, after you issue the AMSERV command. To indicate to AMSERV that you are using tape for input or output, you must use the TAPIN or TAPOUT option to specify the tape device being used:

```
labeldef tapedd fid filename...
amserv export (tapout 181
```

In this example, the output from an EXPORT function is to a tape at virtual address 181. CMS prompts you to enter the ddname:

```
DMSAMS367R ENTER TAPE OUTPUT DDNAMES:
```

After you enter the ddname (TAPEDD in this example) for the tape file, AMSERV begins execution.

AMSERV in CMS assumes that tape volumes used for input and/or output have IBM standard tape labels, i.e., VOL1, HDR1, etc. The user can override this default by indicating to AMSERV via Access Method Services control statements to use non label tapes. If standard label tapes are used the LABELDEF command is required. The CMS/DOS routine that performs the tape open needs label information for standard label tapes. See the description of the LABELDEF command in Chapter 6, "Using Real Printers, Punches, Readers, and Tapes" on page 6-1 for further information. The filename you specify on the LABELDEF command should be the same one you use to reply to the access method service message that requested you to supply the tape's ddnames. However, the LABELDEF command must be issued before the AMSERV command. If you only want the tape labels skipped, but not checked, enter a LABELDEF with no parameters other than filename.

Standard label tapes used for input must always contain standard VOL1, HDR1, and EOF1 labels or they are rejected by CMS AMSERV. Standard label output tapes do not need to contain VOL1 labels because the user is prompted to enter a volume serial number and have the VOL1 label written if he wants. However, blank tapes should not be used for output because the open routine tries to read the tape.

Reading Tapes

When you create a tape file using AMSERV under CMS, CMS writes a mark preceding each output file. When CMS AMSERV is used to read this same file, it automatically skips past the tape mark to read the file. If you want to read the tape on a real OS/VS system, however, you must use the LABEL=SL as a parameter on the data definition (DD) card for the tape. When you create a tape file using AMSERV under CMS, CMS writes a label file preceding each output file. When CMS AMSERV is used to read this same file, it checks the HDR1 and VOL1 labels using the LABELDEF command you provide before it reads the data file. If you want to read the tape on a real OS/VS system, however, you must use the LABEL=SL as a parameter on the data definition (DD) card for the tape. If you want to read the tape on a real OS/VS system, however, you must use either LABEL=SL or LABEL=(2,NL) as a parameter on the data definition (DD) card for the tape.

If you are creating a tape under OS/VS access method services to be read by CMS AMSERV, you must be sure to create the tape using standard labels so that CMS can read it properly. CMS will not be able to read a tape created with LABEL=(,NL) on the DD card.

For CMS to read this tape for any other purpose (for example, to use the MOVEFILE command to copy it), you must remember to forward space the file past the tape mark before beginning to read it.

Using AMSERV Under CMS

This section provides examples of AMSERV functions executed under CMS. The examples are applicable to both the CMS (OS) and CMS/DOS environments. You should be familiar with the material presented in either "Defining DOS Input and Output Files" or "Defining OS Input and Output Files," depending on whether you are a DOS or an OS user, respectively. For the examples shown below, command lines and options that are required only for CMS/DOS users are shaded. OS users should ignore these shaded entries.

A CMS format variable file cannot be used directly as input to AMSERV functions as a variable (V) or variable blocked (VB) file because the standard variable CMS record does not contain the BL and RL headers needed by the variable record modules. If these headers are not included in the record, errors will result.

All files placed on the CMS disk by AMSERV will show a RECFM of V, even if the true format is fixed (F), fixed blocked (FB), undefined (U), variable or variable blocked. The programmer must know the true format of the file he is trying to use with the AMSERV command and access it properly or errors will result.

A CMS standard variable-format file can be accessed as RECFM=U to use the file as follows:

```
AMSERV  AMREPUV
```

The file AMREPUV AMSERV contains the following 2 cards:

```
REPRO  INFILE (INPUT ENV(RECFM(U),BLKSZ(800),PDEV(3330)))
       OUTFILE (OUTPUT ENV(RECFM(V),BLKSZ(800),RECSZ(84),PDEV(3330)))
```

The input file can be any CMS file with LRECL 800 or less. The output file will be a true variable file that can be used with AMSERV.

Using the DEFINE and DELETE Functions

When you use the DEFINE and DELETE control statements of AMSERV, you do not need to specify the DSN parameter on the DLBL command:

```
assgn syscat c
dlbl ijsysct c (perm extent syscat
```

If the above commands are executed prior to an AMSERV command to define a master catalog, the DEFINE will be successful as long as you have assigned a data set name using the NAME parameter in the AMSERV file. The same is true when you define clusters, or when you use the DELETE function to delete a cluster, space, or catalog.

When you do not specify a data set name, AMSERV obtains the name from the AMSERV file. In the case of defining or deleting space, no data set name is needed; the FILE parameter corresponding to the ddname is all that is necessary, and AMSERV assigns a default data set name to the space.

When you define space on a minidisk using AMSERV, CMS does not check the extents you specify to see whether they are greater than the number of cylinders available. As long as the starting cylinder is a valid cylinder number and the

extents you specify are on cylinder boundaries, the DEFINE function completes successfully. However, you receive an error message when you use an AMSERV function that tries to use this space.

Defining a Suballocated Cluster

To define a cluster for VSAM space that has already been allocated, you need:

1. An AMSERV file containing the control statements necessary for defining the cluster, and

2. The master catalog (and, perhaps, user catalog) volume, which will point to the cluster.

The volume on which the cluster is to reside does not have to be online when you define a suballocated cluster.

For example, the file CLUSTER AMSERV contains the following:

```
DEFINE CLUSTER  (  NAME (BOOK.LIST) -
      VOLUMES (123456) -
      TRACKS (40) -
      KEYS (14,0) RECORDSIZE (120,132)  ) -
    DATA (NAME (BOOK.LIST.DATA) ) -
    INDEX (NAME (BOOK.LIST.INDEX) )
```

To execute this file, you would need to enter the following command sequence (assuming that the master catalog, on volume 123456, is in your virtual machine at address 310):

```
access 310 b
assgn syscat b
dlbl ijsysct b (perm syscat
amserv cluster
```

Defining a Unique Cluster

For a unique cluster (one defined with the UNIQUE attribute), you must define the space for the cluster at the same time you define its name and attributes; thus the volume or volumes on which the cluster is to reside must be mounted and accessed when you execute the AMSERV command. You can supply extent information for the cluster's data and index portions separately.

To execute an AMSERV file named UNIQUE which contains the following (the ellipses indicate that the AMSERV file is not complete):

```
DEFINE CLUSTER -
      (NAME (PAYROLL) ) -
      DATA ( FILE (UDATA) -
             UNIQUE -
             VOLUMES (567890) -
             CYLINDERS (40) -
             ... ) -
      INDEX ( FILE (UINDEX) ) -
             UNIQUE -
             VOLUMES (567890) -
             CYLINDERS (10) -
             ... )
```

the command sequence should be:

```
access 350 c
assgn sys004 c
dlbl udata c (extent sys004
DMSDLB331R ENTER EXTENT SPECIFICATIONS:
800 800 c sys004
dlbl uindex c (extent sys004
600 200 c sys004
amserv unique
```

Deleting Clusters, Spaces, and Catalogs

When you use AMSERV to delete a VSAM cluster, the volume containing the cluster does not have to be accessed unless the volume also contains the catalog in which the cluster is defined. In the case of data spaces and user catalogs or the master catalog, the volume(s) must be mounted and accessed in order to delete the space.

When you delete a cluster or a catalog, you do not need to use the DLBL command, except to define the master catalog; AMSERV can obtain the necessary file information from the AMSERV file.

You should be particularly careful when you are using temporary disks with AMSERV, that you have not cataloged a temporary data space or cluster in a permanent catalog. You will not be able to delete the space or cluster from the catalog.

Using the REPRO, IMPORT, and EXPORT (or EXPORTRA/IMPORTRA) functions

You can manipulate VSAM files in CMS with the REPRO, IMPORT, and EXPORT functions of AMSERV. You can create VSAM files from sequential tape or disk files (on OS, DOS, or CMS disks) using the REPRO function. Using REPRO, you can also copy VSAM files into CMS disk files or onto tapes. For the IMPORT/EXPORT process, you have the option (for smaller files) of exporting VSAM files to CMS disks, as well as to tapes.

You cannot, however, use the EXPORT function to write files onto OS or DOS disks. Nor can you use the REPRO function to copy ISAM (indexed sequential) files into VSAM data sets, since CMS cannot read ISAM files.

When creating a VSAM file from a non-VSAM disk file, the device track size must be the maximum BLOCKSIZE in the INFILE statement. AMSERV expects a DOS or OS file as input and will not open a disk file when the BLOCKSIZE specified is greater than the track capacity of the disk device being used.

You cannot use the ERASE or PURGE options of the EXPORT command if you are exporting a VSAM file from a read-only disk. The export operation succeeds, but the listing indicates an error code 184, meaning that the erase function could not be performed.

You should not use an EXPORT DISCONNECT function from a CMS minidisk and try to perform an IMPORT CONNECT function for that data set onto an OS system. OS incorrectly rebuilds the data set control block (DSCB) that indicates how much space is available.

The AMSERV file below gives an example of using the REPRO function to copy a CMS sequential file into a VSAM file. The CMS input file must be sorted in

alphameric sequence before it can be copied into the VSAM file, which is a keyed sequential data set (KSDS). The VSAM cluster, NAME.LIST, is defined in an AMSERV file named PAYROLL:

```
DEFINE CLUSTER ( NAME (NAME.LIST  ) -
       VOLUMES (CMSDEV) -
       TRACKS (20) -
       KEYS (14,0) -
       RECORDSIZE (120,132) ) -
    DATA (NAME (NAME.LIST.DATA) ) -
    INDEX (NAME (NAME.LIST.INDEX ) )
```

To sort the CMS file, create the cluster and copy the CMS file into it, use the following commands:

```
sort name list a name sort a
DMSSRT604R ENTER SORT FIELDS:
1 14
access 135 c
assgn syscat c
dlbl ijsysct c (perm syscat
amserv payroll
assgn sys006 a
dlbl sort a cms name sort (sys006
assgn sys007 c
dlbl name c dsn name list (sys007 vsam
amserv repro
```

The file REPRO AMSERV contains:

```
REPRO   INFILE ( SORT  -
          ENV (RECORDFORMAT (F) -
               BLOCKSIZE (80) -
               PDEV (3330) ) ) -
         OUTFILE (NAME)
```

When you use the REPRO, IMPORT, or EXPORT functions with tape files, you must remember to use the TAPIN and TAPOUT options of the AMSERV command. These options perform two functions: they allow you to specify the device address of the tape, and they notify AMSERV to prompt you to enter a ddname.

In the example below, a VSAM file is being exported to a tape. The file, TEXPORT AMSERV, contains:

```
EXPORT  NAME.LIST -
        INFILE (NAME) -
        OUTFILE (TAPE ENV (PDEV (2400) ) )
```

To execute this AMSERV, you enter the commands as follows:

```
assgn sys006 c
dlbl name c (sys006 vsam
amserv texport (tapout 181
DMSAMS367R ENTER TAPE OUTPUT DDNAMES:
tape
```

The fid, volid, and exdte parameters on LABELDEF are only examples; you can substitute any value you want for them on your tape label.

Writing EXECs for AMSERV and VSAM

You may find it convenient to use EXEC procedures for most of your AMSERV functions, as well as setting up input and output files for program execution, and executing your VSAM programs. If, for example, a particular AMSERV function requires several disks and a number of DLBL statements, you can place all of the required commands in an EXEC file. For example, if the file below is named SETUP EXEC:

```
ACCESS 135 B
ACCESS 136 C
ACCESS 137 D
ACCESS 300 G
ASSGN SYSCAT G
DLBL IJSYSCT G (PERM SYSCAT
ASSGN SYS001 B
DLBL FILE1 B DSN FIRST FILE (VSAM SYS001
ASSGN SYS002 C
DLBL FILE2 C DSN SECOND FILE (VSAM SYS002
ASSGN SYS003 D
DLBL FILE3 D DSN THIRD FILE (VSAM SYS003
AMSERV MULTFILE
```

to invoke this sequence of commands, all you have to enter is the name of the EXEC:

```
setup
```

If you place, at the beginning of the EXEC file, the EXEC control statement:

```
&ERROR &EXIT &RETCODE
```

then, you can be sure that the AMSERV command does not execute unless all of the prior commands completed successfully.

For those AMSERV functions that issue response messages, you can use the &STACK EXEC control statement. For example:

```
&ERROR &EXIT &RETCODE
ACCESS 305 D
ASSGN SYS007 D
DLBL OUTPUT D (VSAM SYS007
LABELDEF TAPE FID FILE1
&ERROR &CONTINUE
&STACK TAPE
AMSERV TIMPORT (TAPIN 181
&IF &RETCODE NE 0 TYPE TIMPORT LISTING
TAPE REW
&EXIT 0
```

When the AMSERV command in the EXEC is executed, the request for the tape ddname is satisfied immediately, by the response stacked with the &STACK statement.

If you are executing a command that accepts multiple response lines, you have to stack a null line as follows:

```
&STACK C SYS002, D SYS003
&STACK
DLBL MULTFILE B (MULT SYS001
```

> **Note:** You can use the &BEGSTACK control statement to stack a series of responses in an EXEC, but you must use &STACK to stack a null line.

VSE/VSAM Macros

The VSE/VSAM macros and their options are supported for use in assembler language programs under CMS/DOS. The VSE/VSAM macros are:

```
ACB             EXLST           SHOWCAT
BLDVRP          GENCB           SHOWCB
DLVRP           MODCB           TCLOSE
ENDREQ          POINT           TESTCB
ERASE           RPL             WRTBFR
```

All options are supported with the exception of "AM=VTAM," which is not supported on any of the macros.

The EXLST EXCPAD exit may be specified but will never be taken in the CMS environment. The reason is that VSE/VSAM takes this exit when it is waiting for I/O to complete, but in the CMS environment I/O is always complete when control is returned to VSE/VSAM.

In addition to the above list of macros, the following list of VSE macros normally used with the VSAM macros are also supported. The following macros are distributed with CMS for use with VSAM only.

VSE macro Supported	**Extent of Support**
CDLOAD	Only supported to the extent required for VSAM execution.
CLOSE	Supported for both VSAM and SAM.
CLOSER	Supported for both VSAM and SAM.
GET	Supported for both VSAM and SAM.
OPEN	Supported for both VSAM and SAM.
OPENR	Supported for both VSAM and SAM.
PUT	Supported for both VSAM and SAM.

Obtaining the VSE/VSAM Macros

The "VSEVSAM EXEC" obtains the VSE/VSAM assembler language macros from the VSE/VSAM Licensed Optional Machine Readable Materials tape, and then creates a VSE/VSAM MACLIB. To install the VSE/VSAM assembler language macros, do the following:

1. Mount the Licensed Optional Machine Readable Materials tape on virtual 181.

2. Load the seven VSE macros (CDLOAD, CLOSE, CLOSER, GET, OPEN, OPENR, and PUT) from the product tape to disk (MAINT 393 is recommended). The actual disk used is not important, as long as the macros are available when VSEVSAM is issued.

3. Issue the CMS VSEVSAM command. Respond to the questions when you are prompted.

The seven VSE macros can be erased from the disk after the maclib is created because the macros will be in the maclib. Once you have created the maclib, you are able to assemble your VSAM assembler applications using the VSE/VSAM assembler macros in the maclib. The VSEVSAM EXEC is documented in the *VM/SP Installation Guide*.

OS/VSAM Macros Supported for Use in CMS

A subset of the OS/VSAM macros are supported for use in CMS. The macros are at an MVS 3.8 level and they are contained in the OSVSAM MACLIB that is shipped with VM/SP. The macros are:

ACB	EXLST	RPL
CHECK	GENCB	SHOWCB
ENDREQ	MODCB	TESTCB
ERASE	POINT	

Some options of the OS/VSAM macros will not work in CMS, since OS/VSAM macro requests are executed using VSE/VSAM code. Figure 11-1 lists the OS/VSAM macros and the supported options.

OS/VSAM Macro	Supported Options	
ACB	AM=VSAM	EXLST=address
	BUFND=number	MAREA=address
	BUFNI=number	MLEN=number
	BUFSP=number	PASSWD=address
	DDNAME=ddname	STRNO=number
	MACRF=ADR, CNV,	
	KEY, NDF, DIR,	
	SEQ, SKP, IN, OUT,	
	NRM|AIX, NRS|RST,	
	NSR, NUB|UBF,	
CHECK	RPL=address	
ENDREQ	RPL=address	
ERASE	RPL=address	
EXLST	AM=VSAM	
	EODAD=address	
	JRNAD=address	
	LERAD=address	
	SYNAD=address	
GENCB BLK=ACB	AM=VSAM	EXLST=address
	BUFND=number	LENGTH=number
	BUFNI=number	MAREA=address
	BUFSP=number	MLEN=number
	COPIES=number	PASSWD=address
	DDNAME=ddname	STRNO=number
	MACRF=ADR, CNV,	WAREA=address
	KEY, NDF, DIR,	
	SEQ, SKP, IN, OUT,	
	NRM|AIX, NRS|RST,	
	NSR, NUB|UBF,	
GENCB BLK=EXLST	AM=VSAM	SYNAD=address
	EODAD=address	COPIES=number
	JRNAD=address	LENGTH=number
	LERAD=address	WAREA=address
GENCB BLK=RPL	ACB=address	ECB=address
	AM=VSAM	KEYLEN=number
	AREA=address	LENGTH=number
	AREALEN=number	NXTRPL=address
	ARG=address	RECLEN=number

Figure 11-1 (Part 1 of 4). Options of OS/VSAM Macros Supported in CMS

OS/VSAM Macro	Supported Options	
	COPIES=number	WAREA=number
	OPTCD=ADR|CNV|KEY,	
	DIR|SEQ|SKP,	
	ARD|LRD, FWD|BWD,	
	ASY|SYN,	
	NSP|NUP|UPD,	
	KEQ|KGE, FKS|GEN,	
	LOC|MVE,	
MODCB ACB	BUFND=number	EXLST=address
	BUFNI=number	MAREA=address
	BUFSP=number	MLEN=address
	DDNAME=ddname	PASSWD=address
	MACRF=ADR, CNV,	STRNO=number
	KEY, NDF, DIR,	
	SEQ, SKP, IN, OUT,	
	NRM|AIX, NRS|RST,	
	NSR, NUB|UBF,	
MODCB EXLST	EODAD=address	
	JRNAD=address	
	LERAD=address	
	SYNAD=address	
MODCB RPL	ACB=address	ECB=address
	AREA=address	KEYLEN=number
	AREALEN=number	NXTRPL=address
	ARG=address	RECLEN=number
	OPTCD=ADR|CNV|KEY,	
	DIR|SEQ|SKP,	
	ARD|LRD, FWD|BWD,	
	ASY|SYN,	
	NSP|NUP|UPD,	
	KEQ|KGE, FKS|GEN,	
	LOC|MVE	
POINT	RPL=address	
RPL	ACB=address	ARG=address
	AM=VSAM	ECB=address
	AREA=address	KEYLEN=number
	AREALEN=number	NXTRPL=address
	OPTCD=ADR|CNV|KEY,	RECLEN=number
	DIR|SEQ|SKP,	
	ARD|LRD, FWD|BWD,	
	ASY|SYN,	
	NSP|NUP|UPD,	
	KEQ|KGE, FKS|GEN,	
	LOC|MVE,	
SHOWCB ACB	AREA=address	OBJECT=DATA|INDEX

Figure 11-1 (Part 2 of 4). Options of OS/VSAM Macros Supported in CMS

OS/VSAM Macro	Supported Options	
	FIELDS=ACBLEN, AVSPAC, BUFND, BUFNI, BUFNO, BUFSP, CINV, DDNAME, ERROR, EXLST, FS, KEYLEN, LRECL, MAREA, MLEN, NCIS, NDELR, NEXCP, NEXT, NINSR, NIXL, NLOGR, NRETR, NSSS, NUPDR, PASSWD, RKP, STMST, STRMAX, STRNO	LENGTH=number
SHOWCB EXLST	AREA=address FIELDS=EODAD, EXLLEN, JRNAD, LERAD, SYNAD	LENGTH=number
SHOWCB RPL	AREA=address FIELDS=ACB, AIXPC, AREA, AREALEN, ARG, ECB, FDBK, FTNCD, KEYLEN, NXTRPL, RBA, RECLEN, RPLLEN	LENGTH=number
TESTCB ACB	ERET=address OBJECT=DATA|INDEX ATRB=UNQ OFLAGS=OPEN OPENOBJ=PATH|BASE|AIX ACBLEN=number AVSPAC=number BUFND=number BUFNI=number BUFNO=number BUFSP=number CINV=number DDNAME=ddname ERROR=number EXLST=address FS=number KEYLEN=number ATRB=ESDS, KSDS, REPL, RRDS, SPAN, SSWD, WCK	LRECL=number MAREA=address MLEN=number NCIS=number NDELR=number NEXCP=number NEXT=number NINSR=number NIXL=number NLOGR=number NRETR=number NSSS=number NUPDR=number PASSWD=address RKP=number STMST=address STRNO=number MACRF=ADR, AIX, CNV, DIR, IN, KEY, NDF, NRM, NRS, NSR, NUB, OUT, RST, SEQ, SKP, UBF
TESTCB EXLST	ERET=address EODAD=address JRNAD=address	LERAD=address SYNAD=address EXLLEN=number
TESTCB RPL	ERET=address AIXFLAG=AIXPKP AIXPC=number FTNCD=number I/O=COMPLETE ACB=address	ARG=address ECB=address FDBK=number KEYLEN=number NXTRPL=address RBA=number

Figure 11-1 (Part 3 of 4). Options of OS/VSAM Macros Supported in CMS

OS/VSAM Macro	Supported Options	
	AREA=address	RECLEN=number
	AREALEN=number	RPLLEN=number
	OPTCD=ADR, ARD,	
	ASY, BWD, CNV,	
	DIR, FKS, FWD,	
	GEN, KEQ, KEY,	
	KGE, LOC, LRD,	
	MVE, NSP, NUP,	
	SEQ, SKP, SYN, UPD	

Figure 11-1 (Part 4 of 4). Options of OS/VSAM Macros Supported in CMS

OS/VSAM Error Codes

Error codes returned by VSE/VSAM in response to OPEN, CLOSE, and Data Management Request macro errors are mapped to the appropriate OS/VSAM error codes.

- Figure 11-2 lists the error codes returned by VSE/VSAM in response to OPEN errors.
- Figure 11-3 on page 11-41 lists the error codes returned by VSE/VSAM in response to CLOSE errors.
- Figure 11-4 on page 11-41 lists the error codes returned by VSE/VSAM in response to Data Management Request macro errors.

If a VSE/VSAM error code cannot be mapped to any OS/VSAM error code, then a CMS error message and an ABEND 35 are issued except for the cases indicated by an "*."

VSE/VSAM Error Code	CMS Error Message or OS/VSAM Error Code	VSE/VSAM Return Code	OS/VSAM Return Code
2	DMSVIP779E	8	8
4	4	8	8
14	DMSVIP782E	8	8
15	DMSVIP782E	8	8
17	DMSVIP782E	8	8
18	DMSVIP782E	8	8
19	DMSVIP782E	8	8
32	DMSVIP782E	8	8
34	DMSVIP782E*	8	8
40	DMSVIP778E	8	8
48	168	8	8
50	DMSVIP782E	8	8
64	188	8	8
65	DMSVIP779E	8	8
66	DMSVIP782E	8	8
67	DMSVIP782E	8	8
68	168	8	8
69	DMSVIP782E	8	8
70	DMSVIP782E	8	8
71	DMSVIP782E	8	8

Figure 11-2 (Part 1 of 3). VSE/VSAM to OS/VSAM Error and Return Code Mapping for OPEN Errors

VSE/VSAM Error Code	CMS Error Message or OS/VSAM Error Code	VSE/VSAM Return Code	OS/VSAM Return Code
72	148	8	8
78	DMSVIP782E	8	8
79	DMSVIP782E	8	8
80	DMSVIP778E	8	8
92	DMSVIP779E	8	8
96	96	4	4
100	100	4	4
104	104	4	4
108	108	4	4
110	160	8	8
113	144	0	4
114	DMSVIP781E	0	4
115	DMSVIP781E	8	8
116	116	4	4
117	DMSVIP782E	8	8
118	0	0	0
128	128	8	8
132	132	8	8
136	136	8	8
144	144	8	8
148	148	8	8
152	152	8	8
160	160	8	8
161	160	8	8
165	DMSVIP782E	8	8
166	DMSVIP782E	8	8
167	DMSVIP782E	8	8
168	168	8	8
180	180	8	8
188	DMSVIP782E	8	8
192	192	8	8
196	196	8	8
212	212	8	8
216	216	8	8

Figure 11-2 (Part 2 of 3). VSE/VSAM to OS/VSAM Error and Return Code Mapping for OPEN Errors

VSE/VSAM Error Code	CMS Error Message or OS/VSAM Error Code	VSE/VSAM Return Code	OS/VSAM Return Code
220	220	8	8
228	228	8	8
232	232	8	8
248	DMSVIP782E	8	8
254	DMSVIP782E	8	8
255	144	8	8

Figure 11-2 (Part 3 of 3). VSE/VSAM to OS/VSAM Error and Return Code Mapping for OPEN Errors

The following table lists the VSE/VSAM to OS/VSAM error code mapping for CLOSE errors.

VSE/VSAM Error Code	CMS Error Message or OS/VSAM Error Code	VSE/VSAM Return Code	OS/VSAM Return Code
2	DMSVIP783E	non-zero	4
4	4	non-zero	4
76	DMSVIP784	non-zero	4
136	136	non-zero	4
144	144	non-zero	4
165	DMSVIP784	non-zero	4
166	DMSVIP784	non-zero	4
167	DMSVIP784	non-zero	4
184	184	non-zero	4
188	0	non-zero	4
228	DMSVIP783	non-zero	4
252	DMSVIP784	non-zero	4
254	DMSVIP784	non-zero	4
255	148	non-zero	4

Figure 11-3. VSE/VSAM to OS/VSAM Error and Return Code Mapping for CLOSE Errors

For Data Management Request errors, all VSE/VSAM error codes are returned to the OS/VSAM user since the VSE/VSAM and OS/VSAM error codes are equivalent, with the following exceptions:

VSE/VSAM Error Code	CMS Error Message or OS/VSAM Error Code	VSE/VSAM Return Code	OS/VSAM Return Code
32	DMSVIP785E	0	0
48	40	8	8
52	Abend 52*	8	8
56	Abend 56*	8	8
128	DMSVIP786E	8	8
208	DMSVIP786E	8	8
212	DMSVIP786E	8	8
216	DMSVIP785E	8	8

Figure 11-4. DATA Management Request Error Return Code Mapping

Chapter 12. Using the CMS Batch Facility

The CMS batch facility provides a way of submitting jobs for batch processing in CMS. You can use the CMS batch facility when:

- You have a job (like an assembly or execution) that takes a lot of time, and you want to be able to use your terminal for other work while the time-consuming job is being run.
- You do not have access to a terminal.

The CMS batch facility is really a virtual machine, generated and controlled by the system operator, who logs on VM/SP using the batch userid and invoking the CMSBATCH command. All jobs submitted for batch processing are spooled to the userid of this virtual machine, which executes the jobs sequentially. To use the CMS batch facility at your location, you must ask the system operator what the userid of the batch virtual machine is.

Submitting Jobs to the CMS Batch Facility

Under a real OS or DOS system, jobs submitted in batch mode are controlled by JCL specifications. Batch jobs submitted to the CMS batch facility are controlled by the control cards /JOB, /SET, and /*, and by CMS commands.

Any application or development program written in a language supported by VM/SP may be executed on the batch facility virtual machine. However, there are restrictions on programs using certain CP and CMS commands, as described later in this section.

Input to the Batch Machine

Input records must be in card-image format, and may be punched on real cards, placed in a CMS file with fixed-length, 80-character records, or punched to your virtual punch. These jobs are sent to the batch virtual machine in one of two ways:

- By reading the real punched card input into the system card reader

  or

- By spooling your virtual punch to the virtual reader of the batch virtual machine.

When you submit a real card deck to the batch machine, the first card in the deck must be a CP ID card. The ID card takes the form:

```
ID  userid
```

where

ID must begin in card column one and be separated from userid (the batch facility virtual machine userid) by one or more blanks.

For example, if your installation's batch virtual machine has a userid of BATCH1, you punch the card:

```
ID BATCH1
```

and place it in front of your deck.

When you are going to submit a job using your virtual punch, you must first be sure that your punch is spooled to the virtual reader of the batch virtual machine:

```
cp spool punch to batch1
```

Submitting Virtual Card Input to the CMS Batch Facility

Virtual card input can be spooled to the batch machine in several ways. You may create a CMS file that contains the input control cards and use the CMS PUNCH command to punch the virtual cards:

```
punch batch jcl (noheader
```

When you punch a file this way, you must use the NOHEADER option of the PUNCH command, since the CMS batch facility cannot interpret the header card that is usually produced by the PUNCH command. As it does with cards in an invalid format, the batch virtual machine would flush the header card.

You can use an EXEC procedure to submit input to the batch machine. From an EXEC, you can punch one line at a time into your virtual punch, using the &PUNCH and &BEGPUNCH EXEC control statements. When you do this, you must remember to use the CP CLOSE command to release the spool punch file when you are finished:

```
CP CLOSE PUNCH
```

If you are using the EXEC to punch individual lines and entire CMS files to be read by the batch virtual machine as one continuous job stream, you must remember to spool your punch accordingly:

```
CP SPOOL PUNCH CONT
&PUNCH /JOB BOSWELL 999888
PUNCH BATCH JCL * (NOHEADER
CP SPOOL PUNCH NOCONT
CP CLOSE PUNCH
```

The /JOB and /* Cards

A /JOB card must precede each job to be executed under the batch facility. It identifies your userid to the batch virtual machine and provides accounting information for the system. It takes the form:

```
/JOB  userid accntnum [jobname] [comments]
```

where :

`userid`
: is your user identification, or the userid under which you want the job submitted. This parameter controls:

1. The userid charged by the CP accounting routines for the system resources used during a job.

2. The name and distribution code that appear on any spooled printer or punch output.

3. The userid to whom status messages are sent while the batch machine is executing the job.

Note: Items 1 and 2 are correct only if the directory for the userid involved contains the accounting option.

`accntnum`
is your account number. This account number appears in the accounting data generated at the end of your job. It overrides the account number in the CP directory entry for the userid specified for this job.

`jobname`
is an optional parameter that specifies the name of the job being run. If you specify a jobname, it appears as the CP spool file identification in the filetype field. The filename field always contains CMSBATCH. See "Batch Facility Output" below.

`comments`
may be any additional information you want to provide.

end of job The /* card indicates the end of a job to the batch facility. It takes the form:

```
/*
```

The batch facility treats all /* cards after the first as null cards. Therefore, if you want to ensure against the previous job not having a /* end-of-job indicator, you should precede your /JOB card with a /* card.

The /* card is also treated as an end-of-file indicator when a file is being read from the input stream. This is a special technique used in submitting source or data files through the card reader and is discussed under "A Batch EXEC for Non-CMS Users."

The /SET Card

The /SET card sets limits on a system's time, printing, and punching resources during the execution of a job. It takes the form:

```
/SET  [TIME seconds] [PRINT lines] [PUNCH cards]
```

where:

`seconds`
is a decimal value that specifies the maximum number of seconds of virtual CPU time a job can use.

`lines`
is a decimal value that specifies the maximum number of lines a job can print.

`cards`
is a decimal number that specifies the maximum number of cards a job can punch.

The default values for the batch facility are set at 32,767 seconds, printed lines, and punched cards per job. Any new limits defined using the /SET card must be less than these maximum settings. The system resources can be set at lesser values than the default values by an installation's system programmer; be sure you know the maximum installation values for batch resource limits before you use the /SET card.

A /SET card can appear anywhere in the job following the /JOB card. The new limits defined by the /SET card apply only to the part of the job that follows the /SET card.

A job can contain up to three /SET cards (one for each operand); a /SET card cannot be entered more than once for the same operand.

Only use /SET cards for the operands whose values you want to change from the default; the default values are reset between jobs. A /SET card for an operand overrides its default but does not reset the other operands.

How the Batch Facility Works

The CMS batch facility, once initialized, runs continuously. When it begins executing a job, it sends a message to the userid of the user submitting the job. If you are logged on when the batch machine begins executing a job that you sent it, you receive the message:

```
MSG FROM BATCHID:  JOB 'yourjob' STARTED
```

When the batch machine finishes processing a job, it sends the message:

```
MSG FROM BATCHID:  JOB 'yourjob' ENDED
```

where yourjob is the jobname you specified on the /JOB card. Before it reads the next job from its card reader, the batch virtual machine:

- Closes all spooling devices and releases spool files
- Resets any spooling devices identified by the CP TAG command
- Detaches any disk devices that were accessed
- Punches accounting information to the system
- Reloads CMS

All of this "housekeeping" is done by the CMS batch facility so that each job that is executed is unaffected by any previous jobs.

If a job that you sent to the batch virtual machine terminates abnormally (abends), the batch machine sends you a message:

```
MSG FROM BATCHID:  JOB 'yourjob' ABEND
```

and spools a CP storage dump of your virtual machine to the printer. The remainder of your job is flushed.

Whenever the batch virtual machine has read and executed all of the jobs in its reader, it waits for more input.

Preparing Jobs for Batch Execution

When you want to submit a job to the CMS batch facility for execution, you should provide the same CMS and CP commands you would use to prepare to execute the same job in your own virtual machine.

You must provide the batch virtual machine with read access to any disk input files that are required for the job. You do this by supplying the LINK and ACCESS command lines necessary. The batch virtual machine has an A-disk (195), so you can enter commands to access your disks as read-only extensions. For example, if you wanted the batch machine to execute a program module named LONDON on your 291 disk, your input file might contain the following:

```
/JOB FISH 012345
CP LINK BOSWELL 291 291 RR SECRET
ACCESS 291 B/A
LONDON
```

Similarly, if you are using the batch virtual machine to execute a program using input and output files, you must supply the file definitions:

```
CP LINK ARDEN 391 391 RR FOREST
ACCESS 391 B/A
FILEDEF INFILE DISK VITAL STAT B
FILEDEF OUTFILE PUNCH
CP SPOOL PUNCH TO BOSWELL
LONDON
```

If you expect printed or punched output from your job, you may need to include the spooling commands necessary to control the output. In the above example, the batch machine's punch is spooled to userid BOSWELL's virtual reader.

Any output printer files produced by your job are spooled by the batch virtual machine to the printer. These files are spooled under your userid and with the distribution code associated with your userid, provided the userid's directory has the accounting option set. You can change the characteristics of these output files with the CP SPOOL command:

```
cp spool e class t
```

di2 refid=cmsbat.controlling spool files If you want output to appear under a name other than your userid, use the FOR operand of the SPOOL command:

```
cp spool e for jonson
```

Output punch files are spooled, by default, to the real system card punch (under your userid), unless you issue a SPOOL command in the batch job to control the virtual card punch of the batch virtual machine.

Restrictions on CP and CMS Commands in Batch Jobs

The batch facility permits the use of many CP and most CMS commands. The following CP commands can be used to control the batch virtual machine:

```
CHANGE          MSG
CLOSE           QUERY
DETACH          REWIND
DUMP            SMSG
DISPLAY         SPOOL
LINK            STORE
LOADVFCB        TAG
```

Notes

1. The CHANGE, CLOSE, and SPOOL commands may not be used to affect the virtual reader.

2. You can not use the detach commmand to detach any spooling devices or the system or IPL disks.

3. The LINK command must be entered on one line in the format:

   ```
   CP LINK userid vaddr vaddr mode password
   ```

   None of the LINK command keywords (AS, PASS, TO) are accepted. If the disk has no password associated with it, you must enter the password as ALL. A maximum of 26 links may be in effect at any one time.

All CP commands in a batch job must be prefaced with the "CP" command.

Since the batch virtual machine reads input from its reader, you cannot use the following commands or operands that affect the reader:

```
ASSGN SYSxxx READER (CMS/DOS only)
DISK LOAD
FILEDEF READER
READCARD
```

The RDCARD macro cannot be used by jobs that run under the CMS batch machine.

Invalid SET command operands are:

```
BLIP        PROTECT
IMPCP       REDTYPE
INPUT       RELPAGE
OUTPUT
```

All of the other CMS SET command operands can be used in a job executing in the batch virtual machine. All forms of the CP SET command are invalid.

Note If the SET TIMER REAL command is used for the batch machine, the timer expires every two seconds (including while batch is waiting for the reader). To avoid this problem, use the command SET TIMER ON.

Batch Facility Output

Any files that you request to have printed during your job's execution are spooled to the real system printer under your userid, unless you have spooled it otherwise. Once released for processing, these output files are under the control of the CP spooling facilities; if you are logged on, you can control the disposition of these files before they are printed with the CLOSE, PURGE, ORDER, and CHANGE commands. See the following section "Purging and Reordering Batch Jobs."

Output files produced by the batch virtual machine are identifiable by the filename CMSBATCH in the CP spool file name field. The spool file type field contains the filetype JOB, unless you specified a jobname on the /JOB card. This applies to both printer and punch output files.

In addition to your regular printed output, the CMS batch facility spools a console sheet that contains a record of all the lines read in, and the responses, error messages, and return codes that resulted from command or program execution. This file is identified by a spool file name of BATCH and a spool file type of CONSOLE.

Purging and Reordering Batch Jobs

When required, you can control the execution of batch virtual machine jobs by purging, reordering, and restarting them; by the same token, because all the closed printer files are queued for system output under the submitting userid, you can change, purge, or reorder these files prior to processing on the system printer.

To purge a job executing under the batch monitor, follow the procedure below:

1. Signal attention and enter the virtual machine environment.
2. Enter the HX (halt execution) Immediate command.
3. Disconnect the virtual machine using the CP DISCONN command.

The HX command causes the batch facility to abnormally terminate. This provides the user with an error message and a CP dump of the batch facility virtual machine. The batch monitor then loads itself again and starts the next job (if any).

To purge an individual input spool file that is not yet executing, issue the CP PURGE command:

```
PURGE READER spoolid
```

In the format above, spoolid is the spool file number of the job to be purged from the batch virtual machine's job queue. For example, the statement:

```
purge reader 123
```

would purge 123 from the batch virtual machine's job queue.

To reorder individual spool files in the batch facility's job queue, use the CP ORDER command:

```
ORDER READER spoolid1 spoolid2...
```

In this format, spoolid1 and spoolid2 are the assigned spool file identifications of the jobs to be reordered.

You can determine which jobs are in the queue by using the CP QUERY command:

```
query reader all
```

This QUERY command lists the filenames and filetypes of all the jobs in the batch virtual machine's job queue. You can then reorder them, using the ORDER command.

Using CMS EXEC Files for Input to the Batch Facility

There are a variety of ways that CMS EXEC procedures can help facilitate the submission of jobs to the CMS batch facility. You can prepare an EXEC file that contains all of the CMS commands you want to execute, and then pass the name of the EXEC to the batch virtual machine. For example, consider the files COPY JCL and COPYF EXEC:

```
COPY JCL:   /JOB CARBON 999999
            EXEC COPYF
            /*

COPYF EXEC: COPYFILE FIRST FILE A SECOND = =
            COPYFILE THIRD FILE A FOURTH = =
```

Then, if you enter the commands:

```
cp spool punch to cmsbatch
punch copy jcl * (noheader
```

the commands in the EXEC file are executed by the batch virtual machine.

You could also use a CMS EXEC to punch input to the batch virtual machine. Using the same commands as in the example above, you might have a CMS EXEC named BATCOPY:

```
CP SPOOL PUNCH TO BATCH3
&PUNCH /JOB CARBON 999999
&PUNCH COPYFILE FIRST FILE A SECOND = =
&PUNCH COPYFILE THIRD FILE A FOURTH = =
&PUNCH /*
CP CLOSE PUNCH
```

Then, when you enter the EXEC name:

```
batcopy
```

the input lines are punched to the batch virtual machine.

The examples above are very simple; you probably would not go to the trouble of sending such a job to the batch virtual machine for processing. The examples do, however, illustrate the two basic ways that you can use CMS EXEC procedures with the batch facility:

1. Invoking a CMS EXEC procedure from a batch virtual machine
2. Using a CMS EXEC procedure to create a job stream for the batch virtual machine

In either case, the EXECs that you use may be very simple or very complicated. In the first instance, an EXEC might contain many steps, with control statements to conditionally control execution, error routines, and so on.

In the second instance, you might have an EXEC that is versatile so that it can be invoked with different arguments so as to satisfy more than one situation. For example, if you want to create a simple CMS EXEC to send jobs to the batch virtual machine to be assembled, it might contain:

```
CP SPOOL PUNCH TO BATCH3 CONT
&PUNCH /JOB ARIEL 888888
&PUNCH CP LINK ARIEL 191 391 RR LINKPASS
&PUNCH ACCESS 391 B/A
&PUNCH ASSEMBLE &1 (PRINT
&PUNCH CP SPOOL PUNCH TO ARIEL
&PUNCH PUNCH &1 TEXT A (NOHEADER
&PUNCH /*
CP SPOOL PUNCH NOCONT CLOSE
```

If this file were named BATCHASM EXEC, then whenever you wanted the CMS batch facility to assemble a source file for you, you would enter:

```
batchasm filename
```

and the batch virtual machine would assemble the source file, print the listing, and send you a copy of the resulting TEXT file.

Sample System Procedures for Batch Execution

To extend the above example a little further, suppose you wanted to process source files in languages other than the assembler language. You want, also, for any user to be able to use this CMS EXEC. You might have a separate EXEC file for each language, and an EXEC to control the submission of the job. This example shows the controlling EXEC file BATCH and the ASSEMBLE EXEC.

```
BATCH EXEC
*    THIS CMS EXEC SUBMITS ASSEMBLIES/COMPILATIONS TO CMS BATCH
*
*    - PUNCH BATCH JOB CARD;
*    - CALL APPROPRIATE LANGUAGE EXEC (&3) TO PUNCH EXECUTABLE COMMANDS
*
     &CONTROL ERROR
     &IF &INDEX GT 2 &SKIP 2
     &TYPE CORRECT FORM IS: BATCH USERID FNAME FTYPE (LANGUAGE)
     &EXIT 100
     &ERROR &GOTO -ERR1
     CP SPOOL D CONT TO BATCHCMS
     &PUNCH /JOB &1 1111 &2
     &PUNCH CP LINK &1 191 291 RR SECRET
     &PUNCH ACCESS 291 B/A
     EXEC &3 &2 &1
     &PUNCH /*
     CP SPOOL D NOCONT
     CP CLOSE D
     CP SPOOL D OFF
     &EXIT
     -ERR1 &EXIT 100

ASSEMBLE EXEC
*    CORRECT FORM IS: ASSEMBLE FNAME USERID
*
*    PUNCH COMMANDS TO:
*      - INVOKE CMS ASSEMBLER
*      - RETURN TEXT DECK TO CALLER
*    &CONTROL ERROR
     &ERROR &GOTO -ERR2
     &PUNCH GLOBAL MACLIB UPLIB CMSLIB OSMACRO
     &PUNCH CP MSG &2 ASMBLING ' &1 '
     &PUNCH ASSEMBLE &1 (PRINT NOTERM)
     &PUNCH CP MSG &2 ASSEMBLY DONE
     &PUNCH CP SPOOL D TO &2 NOCONT
     &PUNCH PUNCH &1 TEXT A1 (NOHEADER)
     &BEGPUNCH
     CP CLOSE D
     CP SPOOL D OFF
     RELEASE 291
     CP DETACH 291
     &END
     &EXIT
     -ERR2 &EXIT 102
```

Executing the Sample CMS EXEC Procedure

If the above CMS EXEC procedure is invoked with the line:

```
batch fay payroll assemble
```

the BATCHCMS virtual machine's reader should contain the following statements (in the same general form as a FIFO console stack):

```
/JOB FAY 1111 PAYROLL
CP LINK FAY 191 291 RR SECRET
ACCESS 291 B/B
GLOBAL MACLIB UPLIB CMSLIB OSMACRO
CP MSG FAY ASMBLING ' PAYROLL '
ASSEMBLE PAYROLL (PRINT NOTERM)
CP MSG FAY ASSEMBLY DONE
CP SPOOL D TO FAY NOCONT
PUNCH PAYROLL TEXT A1 (NOHEADER)
CP CLOSE D
CP SPOOL D OFF
RELEASE 291
CP DETACH 291
/*
```

When the batch facility executes this job, the commands are executed as you see them: if you are logged on, you receive, in addition to the normal messages that the batch facility issues, those messages that are included in the EXEC.

A Batch EXEC for a Non-CMS User

Many installations run the CMS batch facility for non-CMS users to submit particular types of jobs. Usually, a series of CMS EXEC files are stored on the system disk so that each user only needs include a card to invoke the EXEC, which executes the correct CMS commands to process data included with the job stream.

For example, if a non-CMS user wanted to compile FORTRAN source files, the following BATFORT EXEC file could be stored on the system disk:

```
&CONTROL OFF
FILEDEF INMOVE TERM (RECFM F BLOCK 80 LRECL 80
FILEDEF OUTMOVE DISK &1 FORTRAN A1 (RECFM F LRECL 80 BLOCK 80
MOVEFILE
GLOBAL TXTLIB FORTRAN
FORTGI &1 (PRINT)
&FORTRET = &RETCODE
&IF &RETCODE NE 0 &GOTO -EXIT
PUNCH &1 TEXT A1 (NOHEADER)
-EXIT &EXIT &FORTRET
```

To use this EXEC, a non-CMS user might place the following real card deck in the system card reader:

```
ID CMSBATCH
/JOB  JOEUSER 1234 JOB10
BATFORT JOEFORT
.
.
.
source file
.
.
.
/* (end-of-file indicator)
/* (end-of-job indicator)
```

When the batch virtual machine executes this job, it begins reading the EXEC procedure from disk, and executes one line at a time. When it encounters the MOVEFILE command, it begins reading the source file from its card reader (the batch facility interprets a terminal read as a request to read from the card reader). It continues reading until it reaches the end-of-file indicator (the /* card), and then resumes processing the EXEC on the next line following the MOVEFILE command line.

Additional functions may be added to this EXEC procedure, or others may be written and stored on the system disk to provide, for example, a compile, load, and execute facility. These EXEC procedures would allow an installation to accommodate the non-CMS users and maintain common user procedures.

Chapter 13. Debugging Your Program Using VM/SP

Debugging is a critical part of the program development process. When you encounter problems executing application programs or when you want to test new lines of code, you can use a variety of CP and CMS debugging commands and techniques to examine your program while it is executing.

You can interrupt the execution of a program to examine and change your general registers, storage areas, or control words such as the program status word (PSW), and then continue execution. Also, you can trace the execution of a program closely, so you can see where branches are being taken and when supervisor calls or I/O interruptions occur.

In many cases, you may never need to look at a dump of a program to identify a problem.

Preparing to Debug

Before beginning to debug a program, you should have a current program listing for reference. When you use VM/SP to debug a program, you can monitor program execution, instruction by instruction, so you must have an accurate list of instruction addresses and addresses of program storage areas. You can obtain a listing of your program by using the PRINT command to print the LISTING file created by the assembler or compiler. To determine the virtual storage locations of program entry points, use the LOAD MAP file created by the LOAD and INCLUDE commands. If you are a CMS/DOS user, use the linkage editor map produced by the DOSLKED command.

If the program that you are debugging creates printed or punched output and you will be executing the program repeatedly, you may not wish all of the output printed or punched. You should place your printer or punch in a hold status, so that any files spooled to these devices are not released until you specifically request it:

```
cp spool printer hold
cp spool punch hold
```

When you are finished debugging you can use the CP QUERY command to see what files are being held and then you can select which files you may want to purge or release.

When a Program Abends

The most common problem you might encounter is an abnormal termination resulting from a program interruption. When a program running in a CMS virtual machine abnormally terminates (abends), you receive, at your terminal, the message:

```
DMSITP141T exception EXCEPTION OCCURRED AT address IN ROUTINE name
```

and your virtual machine is returned to the CMS environment. From the message you can determine the type of exception (program check, operation, specification, and so on), and, often, the instruction address in your program at which the error occurred.

Sometimes this is enough information for you to correct the error in your source program, recompile it and attempt to execute it again.

When this information does not immediately identify the problem in your program, you can begin debugging procedures using VM/SP. To access your program's storage areas and registers you can enter the command:

```
debug
```

immediately after receiving the abend message. This command places your virtual machine in the debug environment.

To check the contents of general registers 0 through 15, issue the DEBUG subcommand:

```
gpr 0 15
```

If you want to look at only one register, enter:

```
gpr 3
```

You might also wish to check the program status word (PSW). Use the PSW subcommand:

```
psw
```

You can examine storage areas in your program using the X subcommand:

```
X 201AC 20
```

In this example, the subcommand requests a display of 20 bytes, beginning at location 201AC in your program. User programs executed in CMS are always loaded beginning at location X'20000' unless you specify a different address on the LOAD or FETCH command. To identify the virtual address of any instruction in a program, you only need to add 20000 to the hexadecimal instruction address.

Resuming Execution After a Program Check

On occasion, you will be able to determine the cause of a program check and continue the execution of your program. There are DEBUG subcommands you can use to alter your program while it is in storage and resume execution.

If, for example, the error occurred because you had forgotten to initialize a register to contain a zero, you could use the DEBUG subcommand SET to place a zero in the register, and then resume execution with the GO subcommand. You can use the GO subcommand to specify the instruction address at which you want execution to begin:

```
set gpr 11 0000
go 200B0
```

An alternate method of specifying a starting address at which execution is to resume is by using the SET subcommand to change the last word of the PSW:

```
set psw  0  000200B0
go
```

If your program executes successfully, you can then make the necessary changes to your source file, recompile, and continue testing.

Using DEBUG Subcommands to Monitor Program Execution

The preceding examples did not represent a wide range of the possibilities for DEBUG subcommands. Nor do they represent the only way to approach program debugging. Some additional DEBUG subcommands are illustrated below. For complete details in using these subcommands, refer to the *VM/SP CMS Command and Macro Reference.*

When you prepare to debug a program with known problems, or when you are beginning to debug a program for the first time, you might want to stop program execution at various instructions and examine the registers, constants, buffers, and so on. To temporarily stop program execution, use the BREAK subcommand to set breakpoints. You should set breakpoints after you load the program into storage, but before you begin executing it. You can set up to 16 breakpoints at one time. For each breakpoint, you assign a value (id), and an instruction address:

```
load myprog
debug
break 0  20BC0
break 1  20C10
break 2  20D00
```

Then, you can return to CMS and begin execution:

```
return
start
```

When the first breakpoint in this example is encountered, you receive the messages:

```
DEBUG ENTERED.
BREAKPOINT 0 AT 20BC0
```

Then, in the debug environment, use the subcommands GPR, CSW, CAW, PSW, and X to display registers, control words, or storage locations.

You can resume program execution with the GO subcommand:

```
go
```

If, at any time, you decide that you do not want to finish executing your program, but want to return to the CMS environment immediately, you must use the HX subcommand:

```
hx
```

There are three subcommands you can use to exit from the debug environment:

1. RETURN, to return to the CMS environment when DEBUG is entered with the DEBUG command

2. GO, to resume program execution when it has been interrupted by a breakpoint

3. HX, to halt program execution entirely and return to the CMS environment

If you try to leave the debug environment with the wrong subcommand you receive the message:

```
INCORRECT DEBUG EXIT
```

and you have to enter the proper subcommand.

Using Symbols with DEBUG

To simplify the process of debugging in the CMS debug environment, you can use the ORIGIN and DEFINE subcommands. The ORIGIN command allows you to set an instruction location to serve as the base for all the addresses you specify. For example, if you specify:

```
origin  20000
```

then, to refer to your virtual storage location 201BC, you only need to enter:

```
x 1bc
```

By setting the DEBUG origin at your program's base address, you can refer to locations in your program by the virtual storage numbers in the listing, rather than having to compute the actual virtual storage address each time. The current DEBUG origin stays in effect until you set it to a different value or until you reload CMS (with the IPL command).

You can use the DEFINE subcommand to assign symbolic names to storage locations so that you can reference those locations by symbol, rather than by storage address. For example, suppose that during a DEBUG session you will repeatedly be examining three particular storage locations labeled in your program NAME1, NAME2, and NAME3. They are at locations 20EF0, 20EFA, and 20F04. Enter:

```
load nameprog
debug
origin 20000
define name1 EF0  10
define name2 EFA  10
define name3 F04  10
break 1 a04
return
start
```

When the specified breakpoint is encountered, you can examine these storage areas by entering:

```
x name1
x name2
x name3
```

You can also refer to these symbols by name when you use the STORE subcommand:

```
store name2 c4c5c3c5c1e4e5d6c9d9
```

The names you specify do not have to be the same as the labels in the program; you can define any name up to eight characters.

Figure 13-1 summarizes the DEBUG subcommands.

DEBUG Subcommand	Function
BREAK	Stops program execution at the specified breakpoint.
CAW	Displays the contents of the channel address word.
CSW	Displays the contents of the channel status word.
DEFINE	Assigns a symbolic name to the virtual storage address.
DUMP	Dumps the contents of specified virtual storage locations to the virtual spooled printer.
GO	Returns control to your program and starts execution at the specified location.
GPR	Displays the contents of the specified general registers.
HX	Halts execution and returns to the CMS command environment.
ORIGIN	Specifies the base address to be added to locations specified in other DEBUG subcommands.
PSW	Displays the contents of the old program status word.
RETURN	Exits from debug environment to the CMS command environment.
SET	Changes the contents of specified control words or registers.
STORE	Stores up to 12 bytes of information starting at the specified virtual storage location.
X	Examines virtual storage locations.

Figure 13-1. Summary of DEBUG Subcommands

What To Do When Your Program Loops

If, when your program is executing, it seems to be in a loop, you should first verify that it is looping, and then interrupt its execution and either

1. halt it entirely and return to the CMS environment
   or

2. resume its execution at an address outside of the loop.

The first indication of a program loop may be either what seems to be an unreasonably long processing time, or, if you have a blip character defined, an inordinately large number of blips.

You can verify a loop by checking the PSW frequently. If the last word repeatedly contains the same address, it is a fairly good indication that your program is in a loop. You can check the PSW by using the Attention key to enter the CP environment. You are notified by the message:

```
CP
```

that your virtual machine is in the CP environment. You can then use the CP command DISPLAY to examine the PSW:

```
cp display psw
```

and then enter the command BEGIN to resume program execution:

```
cp begin
```

If you are checking for a loop, you might enter both commands on the same line using the logical line end:

```
cp display psw#begin
```

When you have determined that your program is in a loop, you can halt execution using the CMS Immediate command HX. To enter this command, you must press the Attention key once to interrupt program execution, then enter:

```
hx
```

If you want your program to continue executing at an address past the loop, you can use the CP command BEGIN to specify the address at which you want to continue execution:

```
cp begin 20cd0
```

Or, you could use the CP command STORE to change the instruction address in the PSW before entering the BEGIN command:

```
cp store psw 0 20cd0#begin
```

Tracing Program Activity

When your program is in a loop, or when you have a program that takes an unexpected branch, you might need to trace the execution closely to determine at what instruction the program goes astray. There are three commands you can use to do this.

- The PER command is a CP command that allows you to monitor different events in a virtual machine as they occur during program execution.
- The SVCTRACE command is a CMS command which traces all SVCs (supervisor calls) in your program.
- The TRACE command is a CP command which allows you to trace different kinds of information, including supervisor call instructions.

Using the CP PER Command

The CP PER command can be used to trace:

- all instructions,
- all successful branches,
- all register alterations,
- all instructions executed in your virtual machine that alter storage.

The CP PER command has many options that allow you selectivity in choosing which events are to be monitored. Trace output for the CP PER command is always produced *after* the instruction executes. The RANGE option allows CP PER to monitor events that occur as a result of the execution of instructions within a specified range (or ranges). For example, if you wish to monitor all instructions within your program , (assuming that your program is 500 bytes in length), you can issue:

```
per instruct range 20000.500
```

There is no need to use the ADSTOP command first as is the case with CP TRACE. Only instructions in the range 20000-204FF are monitored.

When your program has been loaded and started, you will receive information at your terminal that might look like this:

```
020000 STM 90ECD00C 00DFAC CC=0
```

This line indicates that a STM instruction (located at address 20000) stored the registers beginning at location 00DFAC and that condition code is now 0.

The CP QUERY command with the PER option can be used to determine what events are currently being traced. For example:

```
query per
```

may result in:

```
1 INSTRUCT RANGE 020000-0204FF TERMINAL NORUN
```

If in addition to instructions, you wish to trace instructions that alter registers, enter:

```
per g range 20000.500
```

To see which events you are monitoring (your current traceset), enter:

```
query per
```

You will see the following:

```
1 INSTRUCT RANGE 020000-0204FF TERMINAL NORUN
2 G RANGE 020000-0204FF TERMINAL NORUN
```

If you continue program execution by entering BEGIN you will receive information at your terminal that might look like this:

```
020004 BALR 05C0   000000 CC=0 G12=40020006
```

This line indicates a BALR instruction at address 020004 changed register 12 to 40020006.

As with CP TRACE, you can specify the printer and/or run for any event. However, CP PER has additional options that can be used with all events.

- The RANGE or FROM option can be used to set up multiple instruction address ranges. This can increase the selectivity with which instruction execution is monitored.
- The PASS option allows you to suppress a specific number of events between displays.
- The CMD option can be used if you want to execute CP command(s) whenever a given event occurs.
- The STEP option can be used to permit a specified number of events to be displayed before the CP command environment is entered.

For more information on the CP PER command, see the *CP Command Reference for General Users*.

Using the CP TRACE Command

You can trace the following kinds of activity in a program using the CP TRACE command:

- Instructions
- Branches
- Interrupts (including program, external,
- I/O and SVC interrupts)
- I/O and channel activity

When the TRACE command executes, it traces all your virtual machine's activity; when your program issues a supervisor call, or calls any CMS routine, the TRACE continues.

You can make most efficient use of the TRACE command by starting the trace at a specific instruction location. You should set an address stop for the location. For example, if you are going to execute a program and you want to trace all of the branches made, you would enter the following sequence of commands to begin executing the program and start the trace:

```
load progress
cp adstop 20004
start
ADSTOP AT 20004
cp trace branch
cp begin
```

Now, whenever your program executes a branch instruction, you receive information at the terminal that might look like this:

```
02001E BALR  05E6     ==> 020092
```

This line indicates that the instruction at address 2001E resulted in a branch to the address 020092. When this information is displayed, your virtual machine is placed in the CP environment, and you must use the BEGIN command to continue execution:

```
cp begin
```

When you locate the branch that caused the problem in your program, you should terminate tracing activity by entering:

```
cp trace end
```

and then you can use CP commands to continue debugging or you can use the EXTERNAL command to cause an external interruption that places your virtual machine in the debug environment:

```
cp external
```

You receive the message:

```
DEBUG ENTERED.
EXTERNAL INTERRUPT
```

And you can use the DEBUG subcommands to investigate the status of your program.

Controlling a CP Trace

There are several things you can do to control the amount of information you receive when you are using the TRACE command, and how it is received. For example, if you do not want program execution to halt every time a trace output message is issued, you can use the RUN option:

```
cp trace svc run
```

Then, you can halt execution by pressing the Attention key when the interruption you are waiting for occurs. You should use this option if you do not want to halt execution at all, but merely want to watch what is happening in your program.

Similarly, if you do not require your trace output immediately, you can specify that it be directed to the printer, so that your terminal does not receive any information at all:

```
cp trace inst printer
```

When you direct trace output to a printer, the trace output is mixed in with any printed program output. If you want trace output separated from other printed output, use the CP DEFINE command to define a second printer at a virtual address lower than that of your printer at 00E. For example:

```
cp define printer 006
```

Then, trace output will be in a separate spool file. CMS printed output always goes to the printer at address 00E.

When you finish tracing, use the CP CLOSE command to close the virtual printer file:

```
cp close e
 -- or --
cp close 006
```

If you want trace output at the printer and at the terminal, you can use the BOTH option:

```
cp trace all both
```

Suspending Tracing

If you are debugging a program that does a lot of I/O, or that issues many SVCs, and you are tracing instructions or branches, you might not wish to have tracing in effect when the supervisor or I/O routine has control. When you notice that addresses being traced are not in your program, you can enter:

```
cp trace end
```

and then set an address stop at the location in your program that receives control when the supervisor or I/O routine has completed:

```
cp adstop 20688
begin
```

Then, when this address is encountered, you can re-enter the CP TRACE command.

Using the SVCTRACE command

If your program issues many SVCs, you may not get all of the information you need using the CP TRACE command. The SVCTRACE command is a CMS command, which provides more detailed information about all SVCs in your program, including register contents before and after the SVC, the name of the called routine, and the location from which it was called, and the contents of the parameter list passed to the SVC.

The SVCTRACE command has only two operands, ON and OFF, to begin and end tracing. SVCTRACE information can be directed only to the printer, so you do not receive trace information at the terminal.

Since the SVCTRACE command can only be entered from the CMS environment, you must use the Immediate commands SO (suspend tracing) or HO (halt tracing) if you want tracing to stop while a program is executing. Use the Immediate command RO to resume tracing.

Since the CMS system is "SVC-driven," this debugging technique can be useful, especially, when you are debugging CMS programs. For more information on writing programs to execute in CMS, see Chapter 8, "Programming for The CMS Environment."

Using CP Debugging Commands

In addition to the CMS debugging facilities, there are CP commands that you can use to debug your programs. These commands are:

- DISPLAY, which you can use to examine virtual storage, registers, or control words, like the PSW
- ADSTOP, which you can use to set an instruction address stop in your program
- STORE, which you can use to change the contents of a storage location, register, or control word

When you use the display command, you can request an EBCDIC translation of the display by prefacing the location you want displayed with a "T."

```
cp display t20000.10
```

This command requests a display of X'10' (16) bytes beginning at location X'20000'. The display is formatted four words to a line, with EBCDIC translation at the left, much as you would see it in a dump.

You can also use the DISPLAY command to examine the general registers. For example, the commands:

```
cp display g
cp display g1
cp display g2-5
```

result in displays of all the general registers, of general register 1, and of a range of registers 2 through 5.

The DISPLAY command also displays the PSW, CAW, and CSW:

```
cp display psw
cp display caw
cp display csw
```

With the STORE command, you can change the contents of registers, storage areas, or the PSW.

As you can see, the CMS DEBUG subcommands and the CP commands ADSTOP, DISPLAY, and STORE, have many duplicate functions. The environment you choose to work in, CP or debug, is a matter of personal preference. The differences are summarized in Figure 13-2. What you should be aware of, however, is that you should never attempt to use a combination of CP commands and DEBUG subcommands when you are debugging a program. Since DEBUG itself is a program, when it is running (that is, when you are in the debug environment), the registers that CP recognizes as your virtual machine's registers are actually the registers being used by DEBUG. DEBUG saves your program's registers and PSW and keeps them in a special save area. Therefore, if you enter the DEBUG and CP commands to display registers, you will see that the register contents are different:

```
gpr 0 15
#cp d g
```

Debugging with CP After a Program Check

When a program that is executing under CMS abends because of a program check, the DEBUG routine is in control and saves your program's registers, so that if you want to begin debugging, you must use the DEBUG command to enter the debug environment.

You can prevent DEBUG from gaining control when a program interruption occurs by setting the wait bit in the program new PSW:

```
cp trace prog norun
```

You should do this before you begin executing your program. Then, if a program check occurs during execution, when CP tries to load the program new PSW, the wait bit forces CP into a disabled wait state and you receive the message:

```
DMKDSP450W CP ENTERED; DISABLED WAIT PSW
```

All of your program's registers and storage areas remain exactly as they were when program interruption occurred. The PSW that was in effect when your program was interrupted is in the program old PSW, at location X'28'. Use the DISPLAY command to examine its contents:

```
cp display 28.8
```

The program new PSW, or the PSW you see if you enter the command DISPLAY PSW, contains the address of the DEBUG routine.

If, after using CP to examine your registers and storage areas, you can recover from the problem, you must use the STORE command to restore the PSW, specifying the address of the instruction just before the one indicated at location X'28'. For example, if the instruction address in your program is X'566' enter:

```
cp store psw 0 20566
cp begin
```

In this example, setting the first word of the PSW to 0 turns the wait bit off, so that execution can resume.

Program Dumps

When a program you execute under CMS abnormally terminates, you do not automatically receive a program dump. If, after attempting to use CMS and CP to debug interactively, you still have not discovered the problem, you may want to obtain a dump. You might also want to obtain a dump if you find that you are displaying large amounts of information, which is not practical on a terminal.

Depending on whether you are using CMS DEBUG or CP to do your debugging, you can use the DUMP command to specify storage locations you want printed. The formats of the DUMP command (CP) and the DUMP subcommand (DEBUG) are a little different. See *VM/SP CMS Command and Macro Reference* for a discussion of the DEBUG subcommand, DUMP; see *VM/SP CP Command Reference for General Users* for a discussion of the CP DUMP command.

In either event, you can selectively dump portions of your virtual storage, your entire virtual storage area, or portions of real storage. For example, in the debug environment, to dump the virtual storage space that contains your program, you would enter:

```
dump 20000 20810
```

The second value depends upon the size of your program.

From the CP environment, enter:

```
cp dump t20000-20810
```

The CP DUMP command allows you to request EBCDIC translation with the hexadecimal dump. The dump produced by the DEBUG subcommand does not provide EBCDIC translation.

Debugging Modules

You can debug nonrelocatable MODULE files (created with the GENMOD command) in the same way you can debug object modules (TEXT files).

To load the MODULE into storage, use the LOADMOD command:

```
loadmod mymod
cp adstop 201C0
start
```

If you make any changes to a module while it is in your virtual storage area, you can generate a new module containing your changes provided your module file includes a load map (created with the MAP option in effect.) When you issue the GENMOD command, the changes become a permanent part of the executable module:

```
loadmod mymod
cp store 201C0 0002
genmod mymod
```

To debug MODULE files in this manner, you must have a listing of the program as it existed when the module was created.

Comparison Of CP And CMS Facilities For Debugging

If you are debugging problems while running CMS, you can choose the CP or CMS debugging tools. Refer to Figure 13-2 for a comparison of the CP and CMS debugging tools.

Function	CP	CMS
Setting address stops.	The CP PER command can be used to set up multiple address stops. The CP ADSTOP command set only one address stop at a time.	Can set up to 16 address stops at a time.
Dumping contents of storage of the printer.	The dump is printed in hexadecimal format with EBCDIC translation. The storage address of the first byte of each line is at the left.	The dump is printed in hexadecimal format. The storage address of the first byte of each line is identified at the left. The contents of general and floating-point registers are printed at the beginning of the dump.
Displaying contents of storage and control registers at the terminal.	The display is typed in hexadecimal format with EBCDIC translation. The CP command displays storage keys, floating-point registers, The display is typed in hexadecimal format.	The CMS commands *do not* display storage key, floating-point registers, or control registers as the CP command does.
Storing information.	The amount of information stored by the CP command is limited only by the length of the input line. The information can be fullword aligned when stored. CP stores data in the floating-point and control registers, as well as in general registers. CP stores data in the PSW, but not in the CAW or CSW. However, data can be stored in the CAW or CSW by specifying the hardware address in the STORE command.	The CMS command stores up to 12 bytes of information. CMS stores data in the general registers, but not in the floating-point or control registers. CMS stores data in the PSW, CAW, and CSW.

Figure 13-2 (Part 1 of 2). Comparison of CP and CMS Facilities for Debugging

Function	CP	CMS
Tracing Information	CP PER provides increased selectivity in tracing the execution of instructions that: • cause successful branches • alter specific storage locations • alter specific general registers • are fetched and executed CP TRACE traces: • All interruptions, instructions and branches. • SVC interruptions • I/O interruptions • Program Interruptions • External interruptions • Privileged instructions • All user I/O operations • Virtual and real CCW's • All instructions The CP tracing is interactive. You can stop it and display other fields.	CMS traces all SVC interruptions. CMS displays the contents of general and floating-point registers before and after a routine is called. The parameter list is recorded before a routine is called.

Figure 13-2 (Part 2 of 2). Comparison of CP and CMS Facilities for Debugging

What Your Virtual Machine Storage Looks Like

Figure 13-3 illustrates a simplified CMS storage map. The portion of storage that is of most concern to you is the user program area, since that is where your programs are loaded and executed. The user program area and some of the other areas of storage shown in the figure are discussed below in general terms.

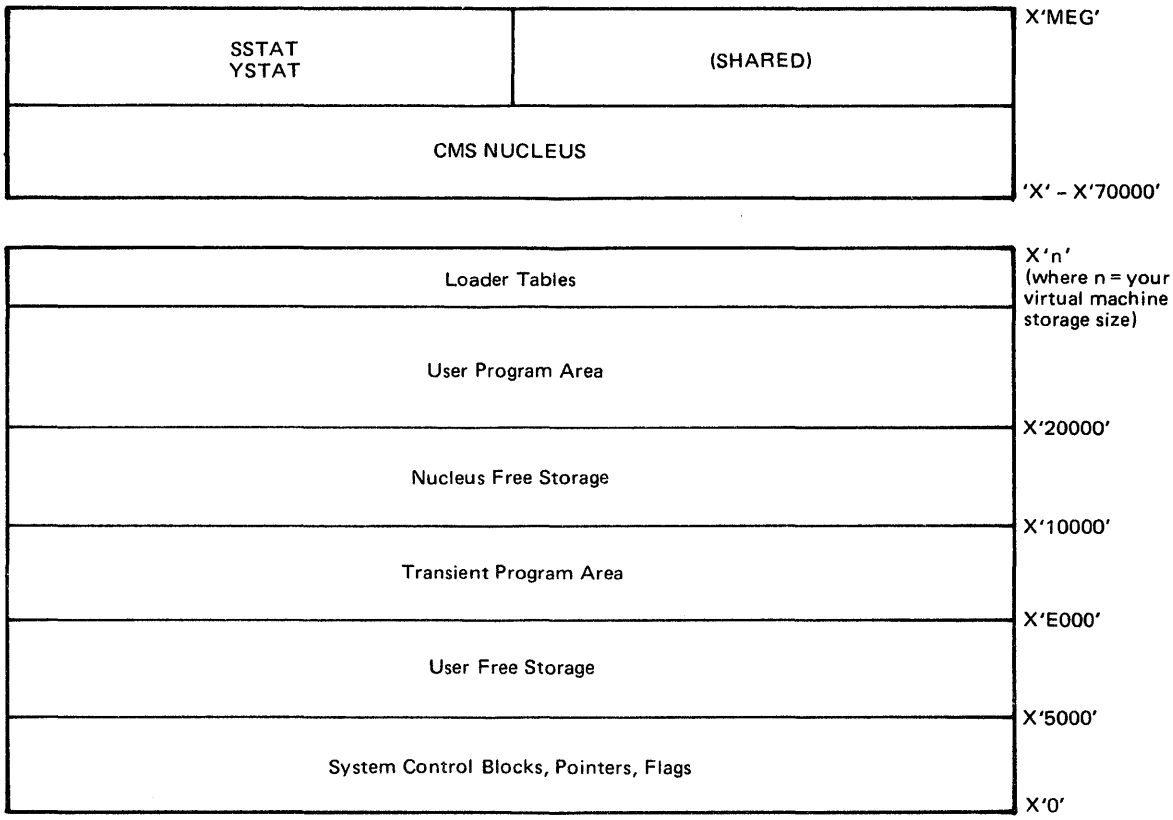


Figure 13-3. Simplified CMS Storage Map

When you issue a LOAD command (for OS or CMS programs) or a FETCH command (for DOS programs), and you do not specify the ORIGIN option, the first, or only, program you load is loaded at location X'20000', the beginning of the user program area.

The upper limit, or maximum size, of the user program area is determined by the storage size of your virtual machine. You can find out how large your virtual machine is by using the CP QUERY command:

```
cp query virtual storage
```

If you need to increase the size of your virtual machine, then you must use the CP command DEFINE. For example:

```
cp define storage 1024k
```

increases the size of your virtual machine to 1024K bytes. If you are in the CMS environment when you enter this command, you receive a message like:

```
STORAGE = 01024K
DMKDSP450W CP ENTERED; DISABLED WAIT PSW '00020000 00000000'
```

and you must reload CMS with the IPL command before you can continue.

You might need to redefine your virtual machine to a larger size if you execute a program that issues many requests for free storage, with the OS GETMAIN or VSE GETVIS macros. CMS allocates this storage from the user program area.

The loader tables, usually located at the top of the user's virtual storage, are used by the CMS loader to point to programs that have been loaded. You can change the size of the CMS loader tables the CMS SET LDRTBLS command. If you use the SET LDRTBLS command, you should issue it immediately after you IPL CMS.

The transient program area is used for loading and executing disk-resident CMS MODULE files that have been created using the ORIGIN TRANS option of the LOAD command, followed by the GENMOD command. For more information on CMS MODULE files and the transient area, see "Executing Program Modules" in Chapter 8, "Programming for The CMS Environment" on page 8-1.

Shared and Nonshared Systems

The areas in storage labeled in Figure 13-3 as the CMS nucleus and the DCSS are system programs that are loaded by various types of requests. When you enter the command:

```
cp ipl cms
```

the area shown as the CMS nucleus is loaded with the CMS system, which is known to CP by its saved name, CMS. This saved system is a copy of the CMS system that is available for many users to share. When you are using CMS, you share it with other users who have also issued the IPL command to load the saved CMS system. By having many users share the same system, CP can manage system resources more efficiently.

Under some circumstances, you may need to load the CMS system into your virtual machine by entering the IPL command as follows:

```
cp ipl 190
```

This IPL command loads the CMS system by referring to its virtual address, which in most installations is 190. The copy of CMS you load this way is nonshared; it is your own copy, but it is the same system, functionally, as the saved system CMS.

Prior to issuing the command 'ipl 190', you must define the size of your virtual machine so that it exceeds the end of the CMS nucleus. This is so that the nucleus can be loaded into your virtual machine.

Some of the CP and CMS debugging commands do not allow you to trace or store information that is contained in shared areas of your virtual machine. For example, if you have entered the command:

```
cp trace inst
```

to trace instructions in your virtual machine, some of the instructions may be located in the CMS nucleus. If you have a shared copy of CMS, you receive a message like:

```
DMKATS181E SHARED SYSTEM CMS REPLACED WITH NONSHARED COPY
```

and CP loads a copy of CMS for you that you do not share with other users.

Discontiguous Saved Segments (DCSS)

Some CMS routines and programs are stored on disks and loaded into storage as needed. These segments include CMS/DOS, VSE SAM, VSAM, and Access Method Services. Beyond the end of your virtual machine address space is an area

of storage into which these segments are loaded when you need them. Since this area is not contiguous with your virtual storage, the segments that are loaded in this area are called discontiguous saved segments.

These segments are loaded only when you need them, and are released from the end of your virtual machine when you are through using them. Like the CMS system, they are saved systems and can be shared by many users. The segments are named CMSDOS (for CMS/DOS), CMSBAM (for VSE SAM interfaces), CMSVSAM (for VSAM interfaces), and CMSAMS (for access method services interfaces). These names are the defaults; they can be changed by the installation.

You can specifically request a nonshared copy of a segment by loading the named system by volume rather than by name. If you do not do this before altering a shared segment (unless with the ADSTOP, TRACE, or STORE CP commands), CP issues the message DMKVMA456W and places you in console function mode. For additional information on saved systems, discontiguous saved segments, and CMS virtual storage, see the *VM/SP System Programmer's Guide*.

Part 3: Learning to use EXECs

Just as important as the CMS editors are the CMS facilities known as the System Product interpreter, EXEC 2 and CMS EXEC processors or interpreters. Using EXEC files, you can execute many commands and programs by entering a single command line from your terminal; in effect , this is like writing your own CMS commands.

In this part, the EXEC facilities are described in general terms to acquaint you with the expressions used in EXEC files and the basic way that EXECs function.

"Chapter 14. Introduction to the EXEC Processors" presents a survey of the basic characteristics and functions of EXEC facilities available to you.

"Chapter 15. Creating System Product Interpreter EXECs" describes how to create and invoke System Product interpreter EXECs. Sample EXECs are provided for you to try.

"Chapter 16. Creating a PROFILE EXEC" describes how you can tailor your virtual machine.

"Chapter 17. Exchanging Data Between Programs through the Stack."

"Chapter 18. CMS Commands Used Along With EXECs"

Chapter 14. Introduction to the EXEC Processors

Three EXEC processors are available:

- System Product interpreter
- EXEC 2
- CMS EXEC

The System Product interpreter handles System Product interpreter programs, which are written in the Restructured Extended Executor (REXX) language. The EXEC 2 processor handles EXEC 2 programs. The CMS EXEC processor handles CMS EXEC programs. EXEC 2 programs and processing are similar to those of the CMS EXEC. The System Product interpreter programs are *not* similar to those of EXEC 2 or CMS EXEC.

The System Product interpreter

The System Product interpreter is an interpretive command and macro processor. It coexists with the CMS EXEC and EXEC 2 processors. The System Product interpreter is functionally a superset of CMS EXEC and EXEC 2, but it uses a completely different language and syntax. There is no compatibility between System Product interpreter programs and those of CMS EXEC or EXEC 2.

VM/SP differentiates System Product interpreter programs from CMS EXEC or EXEC 2 programs by their first statement. The first statement of every System Product interpreter program must be a comment. A comment begins with a /*, and ends with an */, with anything you want in between. For example:

```
/* This is a comment. */
```

The System Product interpreter functions are easy to learn and use. They use a general-purpose, high-level language called REXX, much like that used by PL/I and other high-level programming languages. REXX instructions use structured programming concepts like IF/THEN/ELSE, SELECT, DO WHILE, etc, which allow you to write programs while using words much like those you use to think and communicate.

Other features of the System Product interpreter and the REXX language are:

- It has a number of useful built-in functions you can use in your programs.
- Programs may be written in mixed case with free form layout (which makes them easier to read and follow).
- It has extensive mathematical capabilities (you can even use it as a desk calculator if you wish).
- There is no limit (except the user's virtual storage size) to the length of manipulated data.
- It is easy to find syntax errors in a program. The System Product interpreter executes programs line-by-line and word-by-word, without translating them to another form (no compiling). Thus, when there's a syntax error, the place where it occurred is clearly indicated.

- You can use the TRACE instruction to see how the System Product interpreter is interpreting a particular instruction. This should help you in debugging.

The following books tell you how to use the System Product interpreter and the REXX language:

- *System Product Interpreter User's Guide* is a step-by-step, tutorial-like, guide to using the System Product interpreter. It is intended for new users. There is also an introductory chapter in the *VM/SP CMS Primer* about the System Product interpreter.

- *System Product Interpreter Reference* is a complete compilation of reference information for using the System Product interpreter. It is intended for all users.

As a CMS user, you should become familiar with the System Product interpreter and use it often to tailor CMS commands to your own needs, as well as to create your own commands.

Complete details about using the System Product interpreter can be found in the books listed above.

The EXEC 2 Processor

The EXEC 2 processor handles EXEC 2 programs. These EXEC 2 programs and processing are similar to CMS EXEC programs and processing.

EXEC 2 differs from CMS EXEC in the following ways:

- There is no 8-byte token restriction. Statements are composed of "words" of up to 255 characters each.

- Commands may be issued from EXEC 2 either to CMS or to specified "subcommand" environments, for example the System Product editor.

- EXEC 2 has extended string manipulation functions.

- EXEC 2 has arithmetic functions for multiplication and division.

- EXEC 2 has extended debugging facilities.

- EXEC 2 supports user defined functions and subroutines.

- EXEC 2 allows CMS user programs to manipulate EXEC 2 variables.

In addition, the EXEC 2 interpreter is used by the System Product editor for XEDIT macro processing support.

Relationship of EXEC 2 and EXEC

EXEC 2 does not support all language keywords and syntax of the CMS EXEC processor. EXEC 2 coexists with the CMS EXEC processor program.

EXEC programs written for the CMS EXEC processor will continue to execute correctly with no user modifications. To run CMS EXEC programs as EXEC 2 programs, you must convert the EXEC programs to EXEC 2 programs. See the publication *VM/SP EXEC 2 Reference* for information on the EXEC 2 language.

You may not use CMS EXEC language statements in an EXEC to be interpreted by the EXEC 2 processor, nor EXEC 2 language statements in an EXEC to be interpreted by the CMS EXEC processor. However, you may call an EXEC 2 procedure from a CMS EXEC procedure, and vice versa. To allow greater user flexibility with EXEC 2 and the System Product interpreter, automatic cleanup of an active OS or VSAM environment is not invoked at command completion as it is in the CMS EXEC processor. It is your responsibility to ensure that OS reset and/or VSAM cleanup functions are invoked when needed. VSAM cleanup can be invoked explicitly by issuing 'DMSVSR' as a command. The CMS EXEC processor invokes both VSAM and OS cleanup after the execution of any CMS command. Consequently, any CMS EXEC invoked resets both the OS and VSAM environments if it contains a CMS command that is executed.

Invoking EXEC 2

EXEC 2 programs may reside in EXEC files (with a filetype of EXEC), and can be invoked by the EXEC 2 interpreter. The EXEC 2 interpreter is invoked in the same way the CMS EXEC interpreter is invoked.

For both CMS EXEC and EXEC 2 files with a filetype of EXEC, CMS examines the first statement of the EXEC file to determine which EXEC processor must handle it. If the first statement of the EXEC is &TRACE, CMS calls the EXEC 2 processor to handle it. If the first statement is not &TRACE or /* a comment */, CMS calls the EXEC processor to handle it.

Note: The &TRACE statement does not have to be the first statement in a file if the file does not have a filetype of EXEC (if the EXEC is invoked by an SVC 202).

Attributes of EXEC 2 Files

EXEC 2 files can have any filename that is valid for a CMS filename. EXEC 2 files have the filetype EXEC for files that are invoked from the CMS environment, and the filetype XEDIT for files used as System Product editor macros.

EXEC 2 files can be either 'F' or 'V' format. The maximum line length for lines read from the console is 130; for lines read from the stack it is 255.

For complete information about EXEC 2, see the publication *VM/SP EXEC 2 Reference.*

The CMS EXEC Processor

A CMS EXEC procedure is a CMS file that contains executable statements. The statements may be CMS or CP commands or EXEC control statements. The execution can be conditionally controlled with additional EXEC statements, or it may contain no EXEC statements at all. In its simplest form, an EXEC file may contain only one record, have no variables, and expect no arguments to be passed to it. In its most complex form, it can contain thousands of records and may resemble a program written in a high-level programming language.

Two CMS commands create EXEC files. One is LISTFILE, which can be invoked with the EXEC option; it creates a file named CMS EXEC. The uses of CMS EXEC files are discussed in Appendix B, "The CMS EXEC Processor"under the heading "CMS EXECs and How To Use Them." The CMS/DOS command, LISTIO, creates an EXEC file named $LISTIO EXEC, which creates records for each of the system and programmer logical unit assignments. The LISTIO command and the $LISTIO EXEC are described in Chapter 10, "Developing VSE Programs Under CMS."

More information on the CMS EXEC facility is found in Appendix B, "The CMS EXEC Processor."

Chapter 15. Creating System Product Interpreter EXECs

Creating a System Product Interpreter EXEC

A System Product interpreter file, like a CMS EXEC or EXEC 2 file, has a filetype of EXEC. To determine which EXEC interpreter will be invoked by an EXEC file, look at the first line in the file.

First line	Intepreter
/* a comment */	System Product interpreter
&TRACE	EXEC 2 Processor
Anything else.	CMS EXEC Facility

You can create EXEC files with the CMS editors, by punching cards, or by using CMS commands or programs. When you create a file (filetype of EXEC) using XEDIT, records are, by default, variable-length with a logical record length (lrecl) of 130 characters and case is upper. The CMS EXEC facility can process variable-length files of up to 130 characters. EXEC 2 can process variable-length files of up to 255 characters. The System Product interpreter processes files of any logical record length (lrecl). For example, to create an EXEC file, enter:

```
xedit new exec
```

If you have a fixed-length file that you want to convert to a variable-length file, then you can edit the EXEC file and issue the XEDIT subcommand:

```
recfm v
```

Or, you can use the COPYFILE command:

```
copyfile new exec a (recfm v
```

Whenever possible, you should use variable-length EXEC files.

If you use XEDIT to create a CMS EXEC or an EXEC 2 EXEC, you cannot enter the EXEC statements in mixed case. Use the XEDIT subcommand:

```
set case uppercase
```

Invoking Your EXEC Files

EXEC procedures are invoked when you enter the filename of the EXEC file. You can precede the filename on the command line with the CMS command, EXEC. For example:

```
exec test
```

where TEST is the filename of the EXEC file. For example, an EXEC named THANKYOU would be executed when you entered either:

```
exec thankyou
 -- or --
thankyou
```

You must precede the EXEC filename with the EXEC command when:

- You invoke an EXEC from CMS EXECs and EXEC 2 EXECs.
- You invoke an EXEC from REXX with "address command." (The default is "address CMS," which means EXEC need not be specified.)
- You invoke an EXEC from a program.
- You call a System Product interpreter EXEC recursively.
- You have the implied EXEC (IMPEX) function set OFF for your virtual machine.

The implied EXEC (IMPEX) function is controlled by the SET command. It allows you to treat EXEC files as commands so that you only must enter the filename of the EXEC program. The default setting for IMPEX is ON; you almost never need to change it. To find out what the IMPEX setting is, enter:

```
q impex
```

If the response is:

```
IMPEX    = OFF
```

this means that the EXEC command must precede the EXEC filename to invoke an EXEC procedure. To set IMPEX to ON, so that you only need to enter the EXEC filename, enter:

```
set impex on
```

An EXEC procedure having a synonym defined for it can be invoked by its synonym if the implied EXEC (IMPEX) function is on. You may use the synonym for an EXEC program within a System Product interpreter program.

One EXEC file that you never have to specifically invoke is a PROFILE EXEC. It automatically executes after you IPL CMS, when your A-disk is accessed. PROFILE EXECs are discussed in Chapter 16, "Creating a PROFILE EXEC."

Sample System Product Interpreter EXECs

Here are two sample System Product interpreter EXECs to give you some flavor of the language.

The first sample is an EXEC to copy a file from any disk to the user's A-disk. Note that the EXEC uses the required first comment statement as a description of its function.

```
/* This exec copies a file from any disk to the user's A-disk */
arg fn ft fm
if fn = '?' then signal tell
if arg( ) > 3 | arg( ) < 2
then do
   Parse source . . me .
   say 'Invalid command for 'me' exec.'
   exit
   end
if fm  = '' then fm = '*'
copyfile fn ft fm '= = a'
exit rc
tell:
parse source . . me .
say 'This exec,' me', copies the given file to'
say 'the A-disk and passes back the return'
say 'code from copyfile'.
exit 100
```

The second sample sends the file that you specify to the userid that you specify. Note that in System Product interpreter EXECs that you do not need to preface a CP command with CP.

```
/* This exec sends a specific file to a specific user */
parse source . . me .
n = arg()
if  n = 0
  then
    do
      say 'Command is:' me 'user fn ft <fm>'
      exit 100
    end
if  n<3 | n>4
  then
    do
      say 'Invalid' me 'message'
      exit 101
    end
arg user fn ft fm
spool punch to user class a
if  rc ¬= 0
  then
    do
      say user 'is not a valid userid'
      exit 102
    end
if fm = '' then fm = 'A'
punch fn ft fm
retsave = rc
spool punch to '*' class a
if retsave ¬= 0
  then
    do
      say 'Error' retsave 'from punch (while in' me')'
      exit 103
    end
msg user 'I have punched you my file' fn ft fm
exit
```

Complete details about the System Product interpreter can be found in the *System Product Interpreter User's Guide* and in the *System Product Interpreter Reference.*

Chapter 16. Creating a PROFILE EXEC

A PROFILE EXEC is different from other EXECs. It has the special filename PROFILE and it is executed automatically whenever you issue "IPL CMS" (or if you have automatic IPL). Your PROFILE EXEC contains the CP and CMS commands that you issue at the start of every terminal session. You can write your PROFILE EXEC for any of the EXEC interpreters. It usually contains commands that:

- access disks.
- describe your terminal and printer.
- set up your PF keys.
- describe macro and text libraries that you commonly use.
- set your screen colors (color terminals only).
- invoke your synonym table.

A PROFILE EXEC written with System Product interpreter statements might look like this:

```
/* sample profile */
access 497 b                    /* ACCESS B DISK               */
set rdymsg smsg                 /* SHORT FORM OF READY MSG     */
set blip '*'                    /* SET BLIP CHARACTER TO *     */
synonym mysyn                   /* INVOKE MY SYNONYM TABLE     */
global maclib osmacro privmac   /* MACRO LIBRARIES             */
global txtlib privlib           /* TEXT LIBRARIES              */
set PF1 immed rdrlist           /* PF1 KEY RDRLIST             */
set PF11 immed filelist         /* PF11 KEY FILELIST           */
set PF12 retrieve               /* PF12 RETRIEVE FUNCTION      */
```

Do not use the CP DEFINE STORAGE command in your PROFILE EXEC. It resets your virtual machine and you would have to IPL CMS again.

You can enter "profile" at any time to execute your PROFILE EXEC. If you make changes to your PROFILE EXEC during your terminal session, the changes will not be in effect until you execute your profile again.

Should you want to suppress the execution of your PROFILE EXEC, the first command you enter after you issue the IPL command is the CMS ACCESS command with the NOPROF option specified. For example, if you enter:

```
ipl cms
```

The system response may be:

```
VM/SP Release 3  03/2/83   10:22:33
```

To suppress the execution of your PROFILE EXEC, you enter:

```
6acc 191 a (noprof
```

When the system responds with:

```
R;
```

you have IPL'ed CMS and accessed your A-disk without invoking your PROFILE EXEC.

You can find more information about the CMS ACCESS command in the *VM/SP CMS Command and Macro Reference.*

Chapter 17. Exchanging Data Between Programs through the Stack

Reading from the Console Stack

When you are in the CMS environment executing programs or CMS commands, you can stack commands, either by entering multiple command lines separated by the logical line end symbol, as follows:

```
print myfile listing#cp query printer
```

or by signaling an attention interruption and entering a command line, as follows:

```
Enter:
print myfile listing
Press the PA1 KEY
Enter:
cp query printer
```

In both of the preceding examples, the second command line is saved in the console stack. Whenever a read occurs in your virtual machine, CMS reads lines from the console stack, if there are any lines in it. If there are no lines in the stack, the read results in a physical read to your terminal (on a typewriter terminal, the keyboard unlocks).

A virtual machine read occurs whenever a command or subcommand finishes execution, or when an EXEC or a program issues a read request. Many CMS commands also issue read requests, for example, SORT and COPYFILE. If you want to execute one of these commands in an EXEC, you may want to stack, in the console stack, the response to the read request so that when it is issued it is immediately satisfied. For example:

```
PUSH "42-121 1 "
COPYFILE &NAME LISTING A '=' ASSEMBLE '= ('SPECS
```

When the COPYFILE command is issued with the SPECS option, a prompting message for a specification list is issued, followed by a read request. In this EXEC, the request is satisfied with the line stacked with the PUSH instruction. If the response were not stacked, you would have to enter the appropriate information from the terminal during the execution of the EXEC that contained this COPYFILE command line.

In addition to stacking predefined responses to commands and programs, you can use the console stack to stack CMS commands and XEDIT subcommands, as well as data lines to be read within the EXEC.

The number of lines that you can place in the console stack at any one time varies according to the amount of storage available in your virtual machine for stacking. You may want to stack one or two lines at a time, or you may wish to stack many lines. There are several features available in the System Product interpreter that can help you manipulate the stack.

Exchanging Data Between Programs through the Stack

The Console Stack is composed of the terminal input buffer and the program stack. Lines typed at the terminal (maximum length of 130 characters per line) are placed

in the terminal input buffer. Lines transmitted by programs through the CMS ATTN function are placed in the program stack (maximum length of 255 characters per line).

When the WAITRD function is called (as a result of a RDTERM macro call, for example), it will look in the most recently created buffer of the program stack (see BUFFER #2 in Figure 17-1). As each buffer is exhausted, RDTERM will look to the next buffer in the program stack (BUFFER #1). If the program stack is empty, WAITRD will then look in the terminal input buffer for an input line. If the terminal input buffer is also empty, then a "console read I/O" will be issued to acquire data from the terminal.

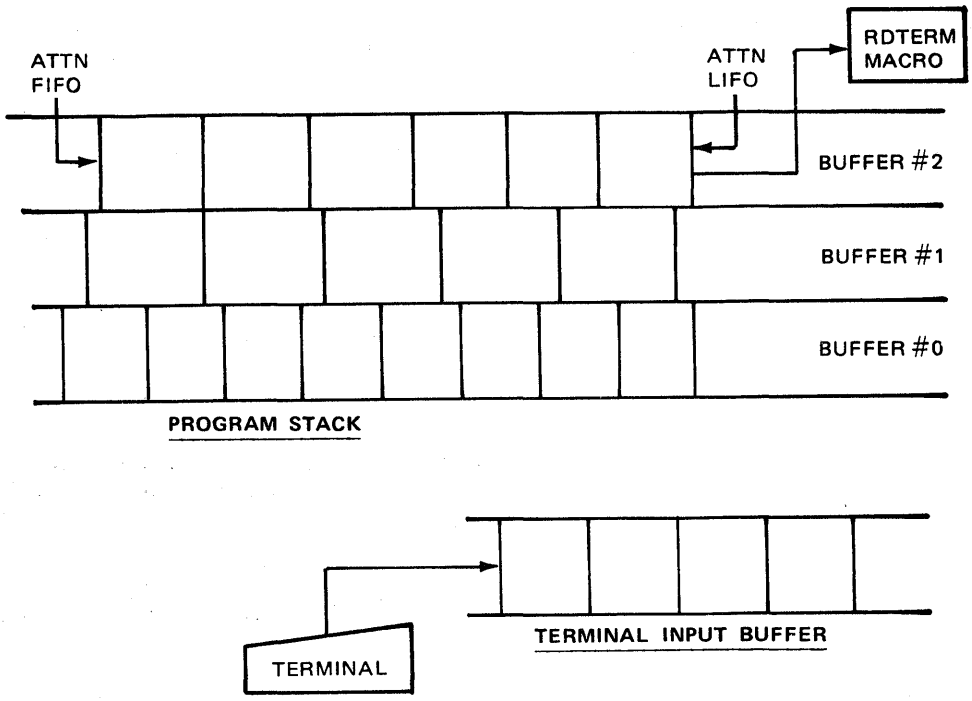


Figure 17-1. The Console Stack

However, when a program issues a RDTERM TYPE=DIRECT, a VM READ is presented at your terminal. The program stack and terminal input buffer are bypassed and unchanged. When the response is entered, the first "logical line" is read and transferred to buffer. If multiple "logical lines" are entered, the remaining lines are added to the terminal input buffer in a FIFO manner.

Previously stacked lines read from the program stack will not have changed since the time they were stored by ATTN (unless uppercase translation has been requested). Before lines are extracted from the terminal input buffer, they are scanned by CP (when typed) for characters defined by the CP TERMINAL command (or for their default values). WAITRD will then scan them for X'15' (logical end of line character), X'00' (physical end of line character), and for any other character defined through a CMS 'SET INPUT' command.

The MAKEBUF, DROPBUF, SENTRIES, and DESBUF CMS commands allow you to create buffers in the program stack, eliminate some or all of the program stack buffers, determine the number of lines in the program stack, and empty both the program stack and the terminal input buffer. These commands may also be

called from a terminal (as CMS commands), from EXEC files, or from assembler language programs. A complete description of these commands can be found in the publication *VM/SP CMS Command and Macro Reference*.

Note: Lines read from the terminal or stacked in the terminal input buffer can be restacked in the program stack, using the ATTN function, and executed at a later time. The line length specified in the parameter list for the ATTN function should be the same length as the line that was previously read from the terminal or the terminal input buffer. A line stacked again by ATTN, using a line length greater than the line length read from the terminal or the console input buffer, may result in an error when execution of the stacked line is attempted.

Chapter 18. Commands Used with System Product Interpreter EXECs

The following CMS commands are used along with a System Product interpreter EXECs. Command formats, descriptions and usage notes for these commands are found in the *VM/SP CMS Command and Macro Reference.*

EXECIO	Manages movement of lines between virtual devices and the program stack. Also causes execution of CP commands and recovers resulting output.
EXECOS	Resets the OS and VSAM environments under CMS without returning to the interactive environment.
GLOBALV	Sets, maintains, and retrieves a collection of named variables.
IDENTIFY	Displays or stacks userid, nodeid, rscsid, date, time, time zone, and day of the week.
IMMCMD	Establishes or cancels Immediate commands from an EXEC.
LISTFILE	Lists information about CMS disk files.
NAMEFIND	Displays/stacks information from a NAMES file (default 'userid NAMES').
QUERY	Requests information about a CMS virtual machine.
RDR	Generates a return code and either displays or stacks a message that identifies the characteristics of the next file in your virtual reader.
SET EXECTRAC	Sets tracing ON or OFF for your System Product interpreter or EXEC 2 EXEC.
XEDIT	Invokes the System Product editor to create or modify a disk file.

The following Immediate commands can be used along with System Product interpreter EXECs.

HI	halt interpretation
TS	trace start
TE	trace end

The following examples show you how you may use some of the CMS commands with your System Product interpreter EXECs.

Using EXECIO

The EXECIO command manages movement of lines between virtual devices and the program stack. It also causes execution of CP commands and recovers resulting output.

This example is not intended to teach you all you need to know to write System Product interpreter programs. If you are not already familiar with the System Product interpreter, see the *VM/SP System Product Interpreter User's Guide*.

The example illustrates how you might use EXECIO commands in a System Product interpreter program to read a CMS file from the program stack, then print that file, 60 lines per page, with the output indented 15 spaces.

This is not the only, nor necessarily the best, way to accomplish the results. However, it does show some uses of the EXECIO command within a System Product interpreter program. The statement numbers in the left margin are to reference explanations below, and are not a part of the program.

Because the program reads, prints, and indents, let's name it RDPRIND EXEC (the filetype must be EXEC).

```
RDPRIND EXEC

1. /* This program reads, prints, and indents */
2. trace n
3. desbuf 0
4. execio 1 print '('cc 1 string
5. arg filename filetype filemode .
6.    do until execiorc ¬=0
7.    execio 100 disk filename filetype filemode
8.    execiorc=rc
9.       do I=1 to queued()
10.       parse pull line
11.       execio 1 print '('string'                    'line
12.       if I//60=0
13.          then execio 1 print '('cc 1 string
14.       end
15.    end
16. close prt name filename filetype filemode
17. exit
```

The following explains the meaning of each statement in the RDPRIND program:

1. The first statement in a System Product interpreter program must always be a comment (/* comment */). Note how we use the comment to tell what the program does.

2. Trace all host commands which return a negative return code.

3. This is a CMS command to clear the program stack (drop all buffers).

4. This is a CMS command to write a line to the printer (space to top of new page).

5. Read in the passed parameters, assigning values to filename, filetype, and filemode. The "." is a placeholder, used here to ignore any passed data after the third parameter.

6. Starts a DO loop. This statement says that the instructions following the DO, up to the END statement which is paired with DO (line 15), should be repeated until the return code from EXECIO (saved in EXECIORC) is not 0.

7. This is a CMS command to read 100 lines from the file called "filename filetype filemode." Those values are set by the ARG command in line 5.

8. The return code from the previous host command (in this case from EXECIO on line 7) is saved in the special variable named RC. This statement saves the return code in a variable called EXECIORC so it can be checked later.

9. Another DO loop starts here, similar to the one started in line 6. In this loop, the set of instructions between the DO and its END (on line 14) will be repeated while I is incremented from 1 until it is equal to the number of lines in the program stack. The System Product interpreter built-in function QUEUED() returns the number of lines in the program stack.

10. This statement reads a line of data from the stack and assigns the variable LINE to the data read.

11. This is a CMS command to write a line to the printer. The blanks will be preserved and the value of LINE (what was read from the stack on line 10) will be placed on the end of the command before it is passed to CMS.

12. This is a conditional check. It asks if the remainder of I divided by 60 is equal to 0. This will be true when I=60, 120, etc.

13. If the previous condition checked (in line 12) is true, then this line is executed. If it's executed, it spaces the printer to the top of a new page (the same command was used in line 4).

14. This END ends the DO loop started in line 9.

15. This END ends the DO loop started in line 6.

16. This is a CP command to close the printer and name the file. Its filename, filetype, and filemode will be set based on the values set in line 5.

17. This statement ends normal processing.

Now, to cause the EXEC to read and print a CMS disk file named TESTFILE DATA A, issue:

```
RDPRIND TESTFILE DATA A
```

TESTFILE, DATA, and A are substituted into the program for filename, filetype, and filemode respectively.

Using EXECOS

The EXECOS command resets the OS and VSAM environments under CMS without returning to the interactive environment. If you request a reset of the OS or VSAM environment, after the execution of a CMS EXEC, the EXECOS command should *precede* the CMS EXEC command. For example:

```
/* example of using EXECOS within an EXEC */
execos exec vmfasm dmsseb dmssp
exit
```

Using GLOBALV

The GLOBALV command sets, maintains, and retrieves a collection of named variables. You can pass these global variables between EXECs.

For example, we have two EXEC files named FIRST EXEC and SECOND EXEC, where the FIRST EXEC calls the SECOND EXEC. The variables are established as global variables in the SECOND EXEC by the statement "globalv put rumors." The statement "globalv get rumors" in the FIRST EXEC assigns the global variables to the FIRST EXEC.

```
/* first exec */
 .
 .
 .
second
 .
globalv get rumors
 .
 .
 .
exit
```

```
/* second exec */
 .
 .
 .
 .
globalv put rumors  /* assign variables */
 .
 .
 .
exit
```

Using IDENTIFY

You can use the information returned by the IDENTIFY command within your EXEC.

For example:

```
/* example of using identify within your exec */
.
.
.
'identify(lifo'   /* get some useful information */
pull userid at locmode via rscsid .
.
.
.
exit
```

Using IMMCMD

The IMMCMD command establishes or cancels Immediate commands from an EXEC.

For the following example we will assume that you have an EXEC that performs a repetitive process. Each time this EXEC is processed, one record is logged to disk. Suppose you wanted to suppress logging of the disk records without terminating the EXEC. Since HX terminates the EXEC, you would not want to use it. Using Pull is not a good alternative since you want to decide at what point to terminate disk logging. You can create your own Immediate command to stop disk logging using the CMS IMMCMD command within your EXEC. For example:

```
/*  Sample EXEC using the CMS IMMCMD command       */
/* Set up stoplog Immediate command               */
IMMCMD SET STOPLOG
/* Set default logging                            */
arg log .
if log='' then log='YES'
if log¬='YES' & log¬='NO' then do
   say 'Invalid parameter :' log
   exit 24
   end
do forever
   /* Check for STOPLOG */
   IMMCMD STATUS STOPLOG
   if rc¬=0 then log='NO'
   /* Perform process ... */
        .
        .
   if log='YES' then EXECIO 1 DISKW LOG FILE A
        .
        .
   end
/* Clear STOPLOG Immediate command */
IMMCMD CLEAR STOPLOG
exit
```

Using LISTFILE

The LISTFILE command lists information about your CMS disk files. You can use this information within your EXEC.

```
/* Example using LISTFILE to find fileid of the first file    */
/* that matches a given filename.                            */
        address command 'MAKEBUF'
        address command 'LISTFILE' filename '* * (FIFO'
        if rc¬=0 then filetype='EXEC'
        else pull filename filetype filemode .
        address command 'DROPBUF'
```

Using NAMEFIND

The NAMEFIND command displays/stacks information from a NAMES file (default 'userid NAMES'). Following is an example of how you can use the CMS NAMEFIND command in an EXEC.

```
/* Program to retrieve phone numbers */
arg nick .
'NAMEFIND :NICK' nick ':PHONE :NAME (LIFO'
if rc¬=0 then do
   say 'Sorry, no phone listing for' nick
   exit
   end
parse pull name
parse pull phone .
if intercom='' then do
   say 'Sorry, no phone listing for' name
   exit
   end
say name'''s phone number is'' phone'.'
exit
```

Using QUERY and RDR

The following example illustrates one way that you can use the information returned from the QUERY and RDR commands in an EXEC.

```
/*  Sample exec to show QUERY and RDR command uses                    */

/*  This section uses the CMS QUERY command to stack information on
    the contents of the users ''A'' disk.  Then, it reads in
    the information (throwing away the header line stacked by
    the QUERY command) and prints out a formatted message.  Unused
    variables set by the PULL command can be displayed if you desire */

query disk a '(' lifo                          /* get disk information */
pull label cuu m stat cyl type,                /* read from the stack, */
     blksize files used '-' percent,           /*   separate into all  */
     left total .                              /*   variables          */
pull .                                         /* read header line     */
used = strip(used,l)                           /* strip leading blanks */
say 'The ''A'' disk is' percent'% full ('used' used blocks out of' total,
    'available)'

/*  This section invokes the CMS RDR command which sets a
    return code depending on the status of the reader and also on
    the type of file in the reader, should one exist.  The System
    Product interpreter sets the variable RC to this returned
    value.  Next, depending on the returned value, this exec
    selectively executes one of several commands.                     */

rdr '('notype                                  /* get info on rdr file  */
                          /* RC set to return code from RDR command */
select
  when rc=0 then say 'Reader is empty'
  when rc=22 then disk load
  when rc=13 then say 'Reader is not ready'
  otherwise
    say 'Return code other than expected'
end
```

Using SET EXECTRAC

You can trace your System Product interpreter or EXEC 2 EXEC by specifying SET EXECTRAC ON prior to EXEC invocation. To turn tracing off, specify SET EXECTRAC OFF.

Using XEDIT

You can use the XEDIT command within an EXEC and stack XEDIT subcommands to manipulate a file.

Writing XEDIT macros

Writing an XEDIT macro is like creating a new XEDIT subcommand. An XEDIT macro is a System Product interpreter or EXEC 2 file invoked from the XEDIT environment.

Refer to the *VM/SP System Product Editor User's Guide* for information on writing XEDIT macros using the System Product interpreter. For information about XEDIT macros written in EXEC 2 language, refer to the *VM/SP EXEC 2 Reference*.

Information about writing CMS EDIT macros using the CMS EXEC facility is found in Appendix B, "The CMS EXEC Processor."

Part 4: The HELP Facility

The CMS HELP facility provides an online display of documentation for CP and CMS messages and commands, System Product interpreter, EXEC 2, and EXEC statements, and XEDIT, EDIT, and DEBUG subcommands. (The System Product editor is invoked by the CMS XEDIT command.)

Chapter 19, "Using the HELP Facility" on page 19-1 describes the HELP Facility, naming conventions for HELP Facility files, and the workings of the HELP Facility.

Chapter 20, "Tailoring the HELP Facility" on page 20-1 describes ways in which you can tailor the HELP Facility to your needs and describes techniques provided by the HELP Facility for creating user HELP description files.

Chapter 19. Using the HELP Facility

The HELP facility uses the System Product editor to display HELP files. The HELP facility is designed for use by 3270-type video terminals in full-screen mode. It can also be used by line-mode terminals.

> **Note:** In some installations, lowercase characters are reserved for display of special alphabets. In such installations, HELP files should be displayed in uppercase representation only. For details, see the *VM/SP Installation Guide*.

The documentation presented by the HELP facility is the same as given in the VM/SP publications. HELP displays the message text, explanation, system action and user action for messages. For commands, HELP will display the description, format, and parameters, or optionally any of these. HELP displays the formats and descriptions for EXEC, EXEC 2 and REXX language statements.

HELP allows you to issue a CP or CMS command directly from the displayed HELP file. Thus, you may issue a command on the command line while viewing the HELP file for that command. The specified command will remain in the command line until you press ENTER, even if you scroll the screen. This feature assists you in remembering what you must specify and how you must specify it.

The HELP facility uses format words similar to those used by the IBM text processor SCRIPT/VS, to build and display files and menus. You must use these format words if you build or alter HELP files. The HELP format words are described in "HELP File Creation" which follows:

If you want to print formatted copies of the HELP files and menus, you need SCRIPT/VS. Refer to "Printing HELP Files" for information on printing methods.

The HELP Facility consists of the following components:

1. CP Commands
2. CMS Commands
3. CP and CMS Messages
4. EDIT Subcommands
5. XEDIT Subcommands
6. DEBUG Subcommands
7. EXEC Statements
8. EXEC 2 Statements
9. Restructured Extended Executor (REXX) language component
10. SQL/Data System Program Product (5748-XXJ) (only if you have this installed on your system.)

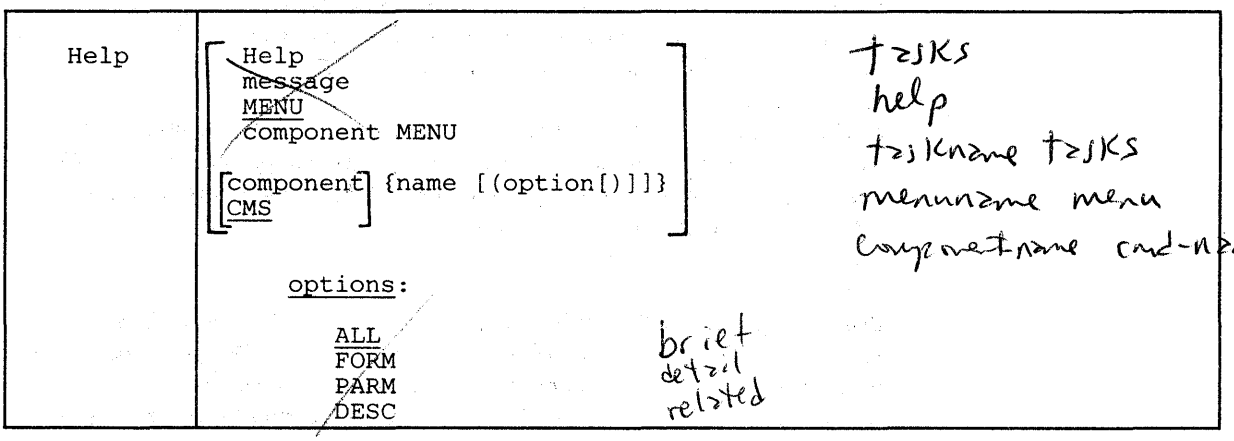
Each of these components (except CP and CMS messages) has a menu that lists all the HELP files available for that component. You may call a HELP file directly or you may call a menu and then select the HELP file from the menu.

If you wish to take advantage of the flexibility of the HELP Facility to tailor the HELP files to fit your own needs, you should also read Chapter 20, "Tailoring the HELP Facility"

Issuing the Help Command

To use the HELP facility, issue the CMS HELP command. The HELP facility allows you to display a menu of the components for which HELP files are available, a menu of the HELP files available for a particular component, and the actual HELP files.

The format of the HELP command is:



where:

HELP
: displays information on how to use the CMS HELP facility. HELP HELP displays a description of the function of the HELP command, its syntax, keywords, operands, and options.

message
: is the 7-character message id you specify to display the HELP file for a message. Specify the message id in the form xxxnnnt, where:

  xxx
  : indicates the component (for example, DMS for CMS messages, DMK for CP messages)

  nnn
  : is the message number

  t
  : is the message type

  Note that you must specify the 7-character message id, not the 10-character id that also identifies the issuing module. For example, specify DMS250S rather than DMSHLP250S for information on that message.

MENU
: displays a list of component menus available. The component menus list the commands, subcommands or EXEC control statements for which HELP files are available. MENU is the default if no parameters are specified.

component
: is the name of the component you want information about. The HELP facility has the following components:

Component	Description
CMS	Conversational Monitor System commands
CP	Control Program commands
DEBUG	CMS DEBUG subcommands
EDIT	CMS EDIT subcommands
EXEC	CMS EXEC statements
EXEC2	EXEC 2 statements
XEDIT	XEDIT subcommands
REXX	System Product interpreter Statements
SQLDS	SQL/Data System Program Product (5748-XXJ) (only if you have this installed on your system.)

`component MENU`
displays the menu of HELP files available for the specified component. There is no default component when you specify component MENU. (For example, if you want to display the menu of CMS commands, you must issue HELP CMS MENU.)

`component name`
displays the HELP file for the specified command, subcommand, or statement. CMS command abbreviations are the only abbreviations allowed when using HELP. If a component is not specified, CMS is assumed. Thus, if you want to display the HELP file for a CMS command, you need only specify:

```
HELP name
```

`option`
allows you to specify DESC, FORM, PARM, or ALL. ALL is the default. The HELP command options are:

ALL
display the specified HELP file starting at the beginning.

DESC
display the specified HELP file starting with the description.

FORM
display the specified HELP file starting with the format specification.

PARM
display the specified HELP file starting with the parameter descriptions.

When a HELP command option is specified, the entire HELP file is made available to the user. The options effect only the initial position of the HELP file display.

Examples:

These are examples of HELP requests issued as CMS commands. Remember that you may also request HELP files directly from menus or from the XEDIT environment.

To request a HELP file for CP message DMK003E, issue:

```
HELP DMK003E
```

To request a menu of CP commands, issue:

```
HELP CP MENU
```

To request a HELP file for the XEDIT LOCATE subcommand, issue:

```
HELP XEDIT LOCATE
```

To request display of the HELP file for the CMS TAPE command beginning with the description, issue:

```
HELP CMS TAPE (DESC or HELP TAPE (DESC
```

Usage Notes

1. If you specify more than one option, only the first is checked for validity.

2. HELP accesses the disk containing the system HELP files, if not already accessed (This disk is specified at system generation time by the system support personnel). The HELP disk is accessed using the first available filemode and remains accessed after HELP has completed processing.

3. For commands or statement names containing one of the following special characters, (for example, EXEC statements &STACK and &END), HELP creates the filename by translating the special character as follows:

   ? is translated to QUESMARK
   = is translated to EQUAL
   / is translated to SLASH
   " is translated to DBLQUOTE
   & is translated to AMPRSAND
   * is translated to ASTERISK
   . is translated to PERIOD

   The first character of the name of the special character replaces the special character in the filename.

   Thus, the statements &STACK and &END would have the filenames ASTACK and AEND. Remember that these changes only apply to the filenames of the statements; they do not affect the way you call for a HELP file display. To display the HELP files for &STACK and &END, you would issue HELP EXEC &STACK and HELP EXEC &END.

   If the name is a single special character, then the filename will be the name of the special character. For example, & and ? have the filenames of AMPRSAND and QUESMARK respectively.

   The following table illustrates these naming conventions.

NAME	FILENAME	CALLED AS
&	AMPRSAND	&
?	QUESMARK	?

NAME	FILENAME	CALLED AS
&STACK	ASTACK	&STACK
&DISK?	ADISKQ	&DISK?
&*	AA	&*
&$	A$	&$

4. Since HELP is a CMS command and uses the CMS ABBREV routine and command tables, abbreviations can only be specified for CMS commands. Abbreviations of commands and subcommands for any "component" other than "CMS" (for example, CP or XEDIT) cannot be resolved.

5. On typewriter terminals, only 12 lines will be displayed and then the user is asked:

   ```
   DO YOU WISH TO CONTINUE? HIT ''ENTER'' (YES) ELSE TYPE ''QUIT''
   ```

   Any entry other than "QUIT" prints 12 more lines unless the end of the file is reached first.

Menus

Menus are alphabetical lists of all HELP files for a component. On terminals having both upper and lower case capability, menus show the minimum abbreviation of a file name you can issue in upper case characters with the remainder of the name in lower case characters (for example, ACcess). See Figure 19-1 on page 19-6 for an example of a displayed menu. You can get a list of all the menus available to you by issuing:

```
HELP or HELP MENU
```

You get a menu by issuing:

```
HELP component MENU.
```

You can request display of a particular HELP file directly from a menu by positioning the cursor at any part of the name and pressing the PF1 key. After the HELP file is displayed, you may return to the menu by pressing PF1 again.

To position the cursor at the file name you want, you can do any one of the following:

- use the key marked --->|, which functions as a tab key, causing the cursor to move to the first character of the next filename.
- use another cursor-movement key.
- type the desired filename and press PF5.

When the cursor is positioned at the desired file name, press the ENTER key or the PF1 key to display the HELP file for that name. The CLOCATE subcommand cannot be used in the MENU files. To find names on MENU screens, enter the desired file name and press PF5.

An asterisk (*) preceding a name in a displayed menu file indicates that the named file itself is a menu file.

```
====> CMS       MENU   <========>  H E L P   I N F O R M A T I O N  <==========
A file may be selected for viewing  by placing the cursor under any character
of the  file wanted  and pressing  the PF  1 or  ENTER key.   A MENU  file is
indicated when  a name is preceded  by an asterisk  (*).  If you are  using a
terminal that  doesn't have a  cursor or PF  keys then  you must type  in the
complete HELP  command with operands  and options.  For a description  of the
operands and options type HELP HELP.


 *DEBUG     CP         EXECIO     HT         MOVEfile   RDRList    SSERV
 *Edit      DDR        EXECOS     HX         NAMEFind   READcard   START
 *EXec      DEBUG      EXECUPDT   IDentify   NAMES      RECEIVE    STATE
 *EXEC2     DEFAULTS   FETch      IMMCMD     NOTE       RELease    SVCtrace
 *REXX      DESBUF     FILedef    INclude    NUCEXT     Rename     SYNonym
 *Xedit     DISK       FILEList   LAbeldef   NUCXDROP   RESERVE    TAPE
 ACcess     DISKID     FINIS      LISTDS     NUCXLOAD   RO         TAPEMAC
 AMserv     DLBL       FORMAT     Listfile   NUCXMAP    RSERV      TAPPDS
 Assemble   DOSLIB     GENDIRT    LISTIO     OPTION     RT         TE
 ASSGN      DOSLKED    Genmod     LKED       OSRUN      RUN        TELL
1= Help      2= Top      3= Quit   4= Return          5= Clocate       6= ?
7= Backward  8= Forward  9= PFKey 10= Backward 1/2  11= Forward 1/2  12= Cursor

====>
                                                          MACRO-READ 2 FILES
```

Figure 19-1 (Part 1 of 2). CMS Menu Display

```
====> CMS       MENU   <========>  H E L P   I N F O R M A T I O N  <==========
 *DEBUG     CP         EXECIO     HT         MOVEfile   RDRList    SSERV
 *Edit      DDR        EXECOS     HX         NAMEFind   READcard   START
 *EXec      DEBUG      EXECUPDT   IDentify   NAMES      RECEIVE    STATE
 *EXEC2     DEFAULTS   FETch      IMMCMD     NOTE       RELease    SVCtrace
 *REXX      DESBUF     FILedef    INclude    NUCEXT     Rename     SYNonym
 *Xedit     DISK       FILEList   LAbeldef   NUCXDROP   RESERVE    TAPE
 ACcess     DISKID     FINIS      LISTDS     NUCXLOAD   RO         TAPEMAC
 AMserv     DLBL       FORMAT     Listfile   NUCXMAP    RSERV      TAPPDS
 Assemble   DOSLIB     GENDIRT    LISTIO     OPTION     RT         TE
 ASSGN      DOSLKED    Genmod     LKED       OSRUN      RUN        TELL
 ATTN       DROPBUF    GLobal     LOAD       PEEK       SENDFile   TS
 CATCHECK   DSERV      GLOBALV    LOADLIB    PRint      SENTRIES   TXTlib
 CMDCALL    Edit       HB         LOADMod    PSERV      SET        Type
 CMSBATCH   ERASE      Help       MACLib     PUnch      SETPRT     Update
 COMpare    ESERV      HI         MAKEBUF    Query      SO         WAITRD
 CONWAIT    EXec       HO         MODmap     RDR        SORT       Xedit
 COPYfile

1= Help      2= Top      3= Quit   4= Return          5= Clocate       6= ?
7= Backward  8= Forward  9= PFKey 10= Backward 1/2  11= Forward 1/2  12= Cursor

====>
                                                          MACRO-READ 2 FILES
```

Figure 19-1 (Part 2 of 2). CMS Menu Display

The System Product Editor

The System Product editor is a full-screen CMS text editor. The HELP facility uses this editor to display HELP files. Many of the features of XEDIT subcommand are available for use on the displayed files. Two of the available features are:

Locate Locate a specified character string in the file or use PF5 to search the file. PF5 positions the cursor under the target string.

Scrolling Move the display up or down.

See the publication *VM/SP System Product Editor Command and Macro Reference* for complete explanations of these features.

Not all features of System Product editor are available for use on the displayed help files. The excluded features are:

FILE	READ
INPUT	SET
MACRO	POWERINP

These are excluded to prevent unnecessary copying of HELP files and to avoid any inadvertent changes to the help files.

While these features will not work on files displayed by the HELP facility, all the System Product editor features are available if you wish to use the XEDIT subcommand to edit the files (calling file by XEDIT filename filetype, not through HELP).

When you issue an XEDIT subcommand to reposition the display of the file on the screen, HELP ensures that a full screen of data is displayed (if there is one). This is done to eliminate blank, or nearly blank, screens.

Using the PA2 Key and the PF Keys

The PA2 key (or its equivalent) and PF keys have the following meanings when using the HELP facility:

Key	Meaning	Usage
PF1	HELP	is used to access HELP files from a menu after the cursor is positioned at the desired file name.
	MENU	is used to return to a menu from a displayed HELP file.
PF2	TOP	moves the display to the top (front) of the HELP file.
PF3	QUIT	goes back to the previous file displayed. (See Figure 19-3 on page 19-10)
PF4	RETURN	exits from displayed HELP file. PF4 quits all HELP files currently in storage. For example, if you call a menu, then called a HELP file from that menu, PF4 quits both the file and the menu and returns control to the originating environment (see Figure 19-3).
PF5	CLOCATE	is the XEDIT subcommand CLOCATE. On the command line, enter the string you are looking for. Then press PF5 to tell HELP to locate the first occurrence of the string, and so on. HELP highlights the line located. For detailed information about the CLOCATE subcommand, refer to the *VM/SP System Product Editor Command and Macro Reference*
PF6	?	displays the last user command issued from the command line.
PF7	BACKWARD	moves the display towards the top of the file one screen. If your screen is 24 lines, then the display is moved up 20 lines.
PF8	FORWARD	moves the display towards the bottom of the file one screen. If your screen is 24 lines, the display is moved down 20 lines.
PF9	PF KEYS	displays a file containing an explanation of PF key meanings for displayed files.
PF10	BACKWARD 1/2	moves the display towards the top of the file one-half a screen. If your screen is 24 lines, the display moves up 10 lines.

Figure 19-2 (Part 1 of 2). Keys in the HELP Facility

Key	Meaning	Usage
PF11	FORWARD 1/2	moves the display toward the bottom of the file one-half a screen. If your screen is 24 lines, the display moves down 10 lines.
PF12	CURSOR	moves the cursor to the command line or to its previous location.
PA2	PRINT	is used with HELP text and menu files. PA2 gives a hardcopy capability. Pressing PA2 while a HELP screen is displayed causes a copy of the current screen to be sent to the currently spooled printer. After quitting HELP, issue CP SP PRT CLOSE to print the file. The PA2 key is set to the XEDIT COPYKEY function (SET PA2 COPYKEY).

Figure 19-2 (Part 2 of 2). Keys in the HELP Facility

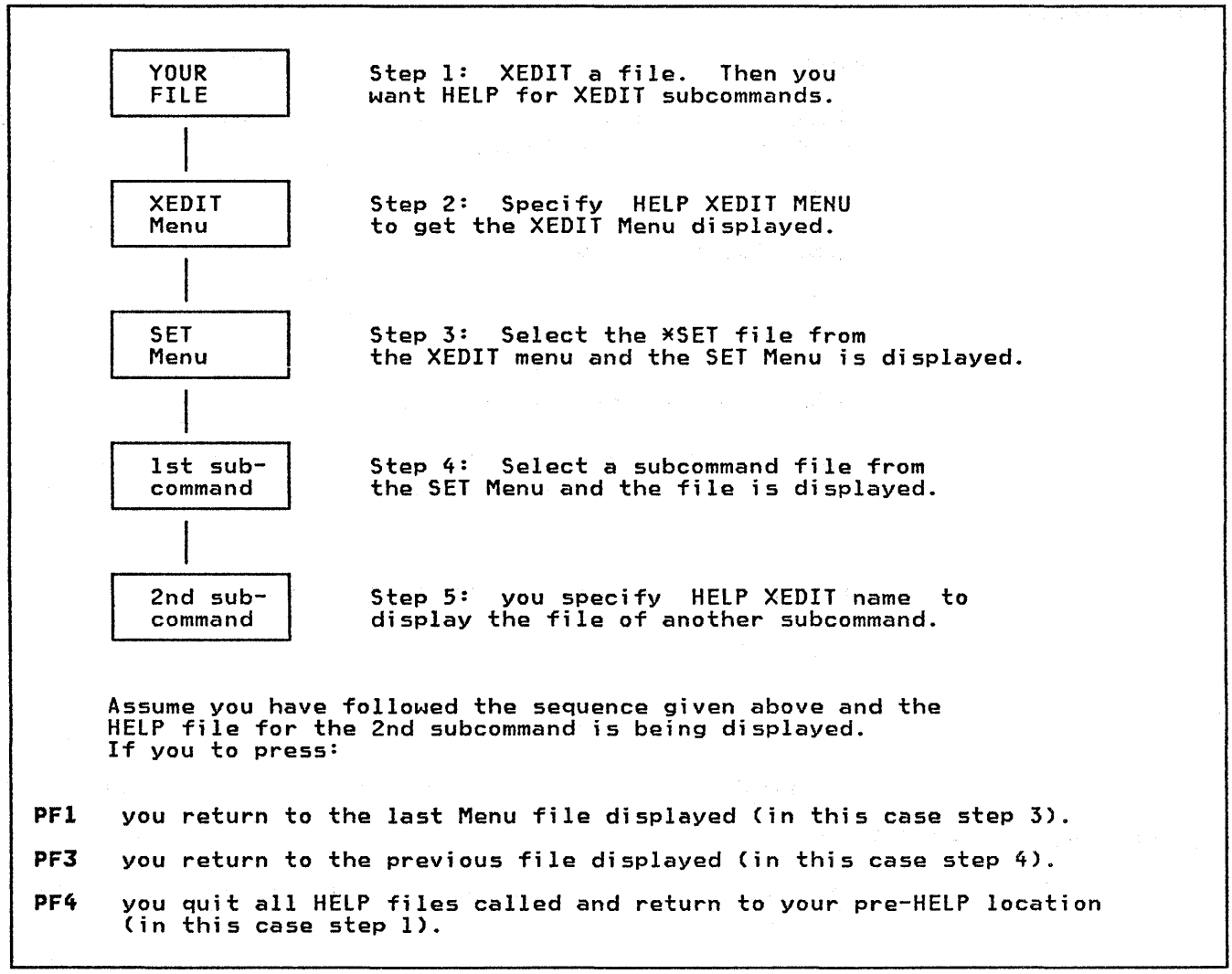


Figure 19-3. Example of Using PF1, PF3, and PF4 in HELP

Printing Help Files

When displaying HELP files, you can get a printed copy of any screen by pressing the PA2 key while the screen is displayed. CMS sends a copy of the displayed screen to the currently spooled printer. This is true for all HELP files.

Remember that all HELP files are CMS files and, as such, can be printed. If you want to print the files formatted, that is, looking as they do when displayed on your screen, you need SCRIPT/VS. If you have this product, you can change the filetype of any file to SCRIPT and then print it like any other SCRIPT file. Without SCRIPT/VS you can only print the HELP files unformatted.

Note: Some special characters used in the HELP files may vary when printed, depending upon the printer used, and the printed output will be continuous with no page breaks.

Notational Conventions

The HELP file notation used to define the command syntax for VM/SP is:

- The use of the less than (<) and greater than (>) Txsymbols denote choices, one of which *must* be selected. For example:

  ```
  <A>
  ```

  indicates that you *must* specify A.

  ```
  <A>
  <B>
  <C>
  ```

  indicates that you *must* specify eitherTx A, B, or C.

- The use of the following notation (shown in the examples) denotes choices, one of which *may* be selected. For example:

  ```
  (A)
  ```

  indicates that you *may* specify A or you may omit the field.

  ```
  + +
  |A|
  |B|
  |C|
  + +
  ```

  indicates that you *may* specify TxA, B, or C, or that you may omit the field.

Naming Conventions for HELP Files

When you extend the HELP text files provided, you must use the following naming conventions for the HELP files:

- The filename for components, commands, subcommands, or EXECs must be the exact full name of the component, command, subcommand, or EXEC.

- The filename for messages has the form xxxnnnt where:

  xxx is the component code prefix (for example, DMS for CMS messages). See *VM/SP System Messages and Codes* for a list of the component code prefixes.

  nnn is the message number.

  t is the message type code (for example, E for error messages in CMS).

  For example, the filename for the CMS message

  ```
  NO FILENAME SPECIFIED
  ```

  would be DMS001E.

- The filetype for components, commands, or EXECs is "HELPxxxx" where xxxx identifies the system associated with this component, command, or EXEC. For example, the filetype for a CMS command would be "HELPCMS."
- The filetype for subcommands is "HELPxxxx" where xxxx identifies the command name associated with this subcommand; for example:

  ```
  HELPDEBU for the DEBUG command.
  ```

- The filetype for messages is "HELPMSG."
- The filetype for a list of all supported commands for a given function is "HELPMENU."

The following illustrates the naming conventions required to interface with the HELP command:

Filename	Filetype	Description
ACCESS	HELPCMS	A CMS command description
EDIT	HELPCMS	A CMS command description
CHANGE	HELPEDIT	An EDIT subcommand description
DMS186W	HELPMSG	A CMS message description
CMS	HELPMENU	A list of the CMS command and/or EXEC names supported by the HELP facility

HELP Facility Filetypes

The filetype of the HELP file is HELPxxxx where xxxx is the name of the component the file belongs to. If the component name is shorter than 4 characters, the filetype is shortened (for example, HELPCP is the filetype for CP commands). If the component name is longer than 4 characters, only the first 4 characters are used (for example, HELPDEBU is the filetype for DEBUG subcommands).

The only exception to the above rule is for EXEC 2 HELP files. Since EXEC and EXEC 2 have the same first four characters, CMS examines the fifth character to determine if the request is for EXEC or EXEC 2. Similarly, since filetypes are limited to 8 characters, CMS assigns the filetype HELPEXEC to EXEC files, and the filetype HELPEXC2 to EXEC 2 files.

Filetypes Reserved for HELP

The reserved filetypes for HELP are:

Filetype	What it is Reserved for
HELPCP	CP commands
HELPCMS	CMS commands
HELPDEBU	DEBUG subcommands
HELPEDIT	EDIT subcommands
HELPEXEC	EXEC statements
HELPEXC2	EXEC 2 statements
HELPHELP	HELP files for HELP
HELPMENU	menus of HELP components
HELPMSG	CMS and CP messages
HELPREXX	System Product interpreter Statements
HELPXEDI	XEDIT subcommands
HELPSET	XEDIT SET Subcommands
HELPPREF	XEDIT PREFIX subcommands
HELPSQLD	SQL/Data System (5748-XXJ) Program Product (only if you have this installed on your system.)

Chapter 20. Tailoring the HELP Facility

One of the most useful features of the HELP facility is its flexibility. You can take full advantage of the CMS file system format used in the HELP facility to tailor it as best suits you.

If you have your own set of HELP files, you can do as you wish with them. However, if you share a set of HELP files with other system users, you will have to get authority from the System Administrator to alter the HELP facility.

What you can do To Your HELP Files

Since all HELP files are CMS files, you can add or delete files or menus, or change any existing file or menu. There are a few restrictions you must follow when tailoring HELP files; they are discussed in the following sections.

> **Note:** If you tailor your HELP files, you should retain documentation of the changes you've made. You can use that documentation to help you update your files when IBM issues update to the HELP facility files.

One way you could do this would be to use the format control word ".cm" indicating that what follows is a comment. For example:

```
.cm HELP format work.
```

HELP will not display any lines in a HELP file that begin with the command ".cm." Thus you could include information about any alterations you have made to your HELP files in the file itself.

Adding HELP Files

You can either add HELP files to existing components or create a new component with its own HELP files.

If you add HELP files to an existing component, you should follow the formatting rules given in Chapter 19, "Using the HELP Facility" for HELP file naming conventions and in this chapter "Creating HELP Files."

If you update a component you should update its menu also. You do this by calling the menu file with the System Product editor, or any other editor, and adding the new names anywhere in the list of names. Remember that the filenames; start in column 1, are one to a line, and are limited to 8 characters.

Deleting HELP Files

You delete HELP files just as you delete any CMS file. Specify ERASE filename filetype to delete a file. If you delete a file, you should delete the filename from the menu for that component also.

Altering Existing HELP Files

To alter a HELP file, first call the file with a text processing editor. Then add or delete as you wish, making sure that you follow the instructions given in "Naming Conventions for HELP Files" on page 19-11 and in"Creating HELP Files" on page 20-3.

Creating Menus for HELP Files

Menus for the HELP facility have the filetype HELPMENU. The filename is the component name they serve (for example, EXEC2 HELPMENU is the filename and filetype for the EXEC 2 menu). Menus contain a list of the HELP files for that component. There are only a few restrictions you must follow when creating menu files. You may precede the list of names with any amount of information for the user. Between this information and the list of names, you must include two lines with the following HELP format words:

```
.sp 2
.fo off
```

Following these commands, you enter the filenames in any order, but they must begin in column 1 of the file, have 8 or less characters, and be one to a line. This list of names is sorted in ascending alphabetical order (in columns 1 thru 8) and is formatted for display on the screen. If there are not two consecutive blank lines found, then the file is considered pre-formatted and is not sorted or formatted by HELP. Any two consecutive blank lines indicates the end of the user information section and the beginning of the list of names for the HELP files for that component. Therefore you are limited to one extra space between items in the user information area.

Example of Menu Creation

Assume you want to add HELP files concerning your internal system procedures to the HELP facility. Chose the component name of SYS4 (System 4) for these procedures. Then create the HELP files for these procedures, giving them a filename and filetype. The filetype should be HELPSYS4. Follow the rules given in "Naming Conventions for HELP Files" on page 19-11. Thus the procedures CLASS8 (a class identifying the type of printing desired) would be a CMS HELP file named "class8 helpsys4."

The menu file for this component would have the filename SYS4 and the filetype HELPMENU and would be set up like below. This menu lists HELP files available for System 4 procedures.

```
.sp 2
.fo off
CLASS8
CLASS7
CLASS0
CLASSC
MOUNT
DEMOUNT
.
.
.
```

When you specify "help sys4 menu," the HELP facility will alphabetize and columnize the filenames and display this file. You may then work with this menu as you would with any other HELP menu.

Changing Menus

If you add, delete or change files, you should change the associated menu. Call the menu file (filename is component name, filetype is HELPMENU) with a editor and make the necessary changes. Remember that there is an eight-character limit on

filenames, only one filename goes on a line, and you can insert filenames anywhere in the list. If you delete a filename, you should delete the line that the filename is on.

Creating HELP Files

The HELP facility enables the user to:

- Extend the command and message description files IBM provides with additional description files of the user's choice
- Produce a formatted terminal display by using the HELP format words when creating the HELP description file.

Creating Additional HELP Files

Users creating additional files for the HELP facility can format their own file or use the format words the HELP facility supports. These format words do the following:

- Draw boxes to enclose tables, illustrations or text
- Place comments within a file
- Indicate that certain input lines are to be included in the formatted output only under certain conditions
- Cause concatenation of input lines and left- and right-justification of output
- Indent only the next input line the specified number of spaces
- Indent a series of input lines the specified number of spaces
- Indent the specified number of spaces all but the first line in a series of input lines
- Insert blank lines between output lines
- Change the final output representation of any input character

The HELP format words are summarized in Figure 20-1 . Descriptions and examples of their use follow.

Format Word	Operand Format	Function	Break	Default Value
.BX (BOX)	V1 V2 ... Vn OFF	Draws horizontal and vertical lines around subsequent output text, in blank columns.	Yes	Draws a horizontal line.
.CM (COMMENT)	Comments	Places comments in a file for future reference.	No	
.CS (CONDITIONAL SECTION)	n ON/OFF	Allows conditional inclusion of input in the formatted output.	No	
.FO (FORMAT -MODE)	ON/OFF	Causes concatenation of input lines, and left and right justification of output.	Yes	On
.IL (INDENT LINES)	n|+n|-n	Indents only the next line the specified number of spaces.	Yes	0
.IN (INDENT)	n|+n|-n	Specifies the number of spaces subsequent text is to be indented.	Yes	0
.OF (OFFSET)	n|+n|-n	Provides a technique for indenting all but the first line of a section.	Yes	0
.SP (SPACE)	n	Specifies the number of blank lines to be inserted before the next output line.	Yes	1
.TR (TRANSLATE)	s t	Specifies the final output representation of any input character.	No	

Figure 20-1. HELP Format Word Summary

Enclosing Text (.BX Format Word)

The HELP facility can insert vertical and horizontal lines in the formatted output to enclose text, illustrations, or tables. You use the .BX format word to specify when you want the horizontal lines to appear and in which columns the vertical lines should appear.

The .BX format word is used in three steps to completely enclose text:

1. The first time you issue the .BX format word, specify the columns in which you want the vertical lines to appear. For example:

   ```
   .bx 1 10 20 30
   ```

   results in the following output:

   ```
   +--------|---------|---------+
   ```

   Note: The first occurrence of the .BX format word causes a horizontal line to appear between the first and last column you specified.

2. After the first issuance of .BX, begin entering the text that is to be enclosed. As HELP formats these lines, vertical lines are placed in the columns that you specified on .BX. However, if a column already has a data character in it, it is not overlaid with the vertical line.

   Note: Whenever you want just a horizontal line to appear (for example, to separate lines in a table), enter the .BX format work without operands. For example:

   ```
   .bx
   ```

   results in the following output:

   ```
            |--------|---------|---------|
   ```

3. When you have finished entering the text that is to be enclosed, issue:

   ```
   .bx off
   ```

   to cause another horizontal line to appear and to prevent any more vertical lines from appearing. This output is:

   ```
   +--------|---------|---------+
   ```

The following example illustrates this technique of enclosing text.

```
.fo off
.bx 1 10 50
.in 2
.of 8
Item 1   Put Item1 text here.
The second line can be written here.
.bx
.of 8
Item 2   Then put Item2 text here.
.bx off
```

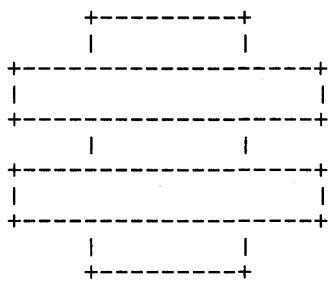
When these input lines are processed, the result is:

```
+--------|--------------------------------------+
| Item1  |Put Item1 text here.                  |
|        |The second line can be written here.  |
|--------|--------------------------------------|
| Item2  |Then put Item2 text here.             |
+--------|--------------------------------------+
```

This example shows how you can change the vertical structure several times in succession. The control words:

```
.bx 10 20
.sp
.bx 5 25
.sp
.bx 10 20
.sp
.bx 5 25
.sp
.bx 10 20
.sp
.bx off
```

result in:



Placing Comments in HELP Files (.CM Format Word)

In addition to text and format words, HELP files can contain comments. Comments are useful for:

- Tracking files. You can include comments that give your name, the date and reason you created a file, and a future date at which the file may be erased.
- Documenting formats. If you use a special format in a HELP file that may be accessed by other people, you may want to place notes within the file explaining how to update the file.
- Place-holders. If a file is incomplete, you may want to put comments in the file where information should be added later.

You can place comments in a HELP file with the .CM format word:

```
.cm Created 10/06/82
.cm Updated 1/3/83
```

HELP does not display comments when processing.

Conditional Display of Text (.CS Format Word)

You can indicate to HELP that certain sections of the file are to be displayed first if the appropriate HELP command options are specified. These options are PARM, FORM, DESC, and ALL. (See *VM/SP CMS Command and Macro Reference* for information on the use of these options.)

In order for HELP command processing to display the appropriate information, you must use the .CS format word in the following manner:

```
.cs 1 on
(text for DESC option)
.cs 1 off
.cs 2 on
(text for FORM option)
.cs 2 off
.cs 3 on
(text for PARM option)
.cs 3 off
```

Use of Format Mode (.FO Format Word)

Format-mode processing means that the HELP facility displays the output lines without breaks, unless specifically requested, and right-justified. You may not want this type of formatting in all cases; you may want certain output to appear exactly as it appears in the HELP file. For this, use the .FO format word to turn off format-mode processing as follows:

```
.fo off
```

When you want to resume format-mode processing, enter:

```
.fo on
```

Format-mode processing is the default.

Indenting Text (.IN and .IL format Words)

When you are creating documents, you may want to set off paragraphs or portions of text by indenting them. This often improves the readability by emphasizing certain text. You can cause paragraphs to be indented using the .IN format word. For example, the lines:

```
This line is not indented.
.in 5
This line is indented.
```

result in:

```
This line is not indented.
     This line is indented.
```

The .IN format word causes a break so that text accumulated before the .IN format word is processed and displayed, then the next text is processed.

The .IN format word effectively sets a new left margin for output text so that when you want text indented you do not have to enter blanks in front of the input lines (as you would for normal typing). HELP continues to concatenate and justify input text lines that begin to column 1, but displays the output indented the number of spaces you specify.

Here's another example:

```
These few lines of text
are formatted
with enough words
.in 5
so that you can
see how HELP's formatting
process
.in +3
continues and may
.in -6
even be reversed, by using a
negative value.
```

These lines result in:

```
These  few  lines  of
text  are   formatted
with enough words
     so that  you can
     see  how  HELP's
     formatting
     process
        continues and
        may
  even  be  reversed,
  by using a negative
  value.
```

In this example, the first .IN format word shifts output to the right five spaces so that text begins in column 6. The second .IN format word requests that the current indentation increase by three spaces so the left margin is now in column 9. When you supply a negative value with the .IN format word, the margin is shifted to the left.

To cancel an indentation that is in effect, you can use a negative value, or you can use the format word:

```
.in 0
```

Because 0 is the default value, you need not specify it when you want to restore the left margin to column 1. You can specify simply:

```
.in
```

When you want to indent only a single line of text (that is, the next output line), use the .IL format word. For example:

```
This line begins in column 1.
.in 5
This line begins in column 6,
which is now the left margin.
.il -3
This line is shifted 3 spaces
to the left of the current margin.
.il 3
This line is shifted 3 spaces to
the right of the current margin.
```

Help processes these lines as follows:

```
This line begins in column 1.
     This   line   begins  in
     column  6, which  is now
     the left margin.
  This   line  is  shifted  3
     spaces  to the  left  of
     the current margin.
        This line  is shifted
     3  spaces  to the  right
     of the current margin.
```

Because the .IL format word causes a break in text, you may find it useful to indicate the beginning of a new paragraph. For example:

```
.il 3
This line begins a paragraph.
.il 3
This line begins another.
```

These lines result in:

```
   This line begins
a paragraph.
   This line begins
another.
```

Use of Offsets (.OF Format Word)

In HELP formatting, an offset differs from an indentation in that offsets do not affect the first line immediately following the format word; the second and

subsequent input lines are indented the specified number of characters. This is useful, for example, when formatting numbered lists where text is blocked to the right of the number.

When a .OF format word is processed, the next text line is printed at the current left margin and subsequent lines (until the next .OF or .IN format word) are offset the specified number of characters. For example, the lines:

```
.of 5
-----This line begins
a 5-character offset.
.of 5
-----This is another line offset
5 characters.
.in 5
An indent changes the left
margin and cancels the offset.
.of 3
---This paragraph begins
at the new left margin.
.of 3
---Here's one more line.
```

result in:

```
-----This  line begins  a
     5-character  offset.
-----This is another line
     offset 5 characters.
     An  indent   changes
     the left  margin and
     cancels the  offset.
     ---This    paragraph
        begins at the new
        left margin.
     ---Here's  one  more
        line.
```

An offset can be canceled with the format word.

```
.of 0
```

This format word causes a break; subsequent text is printed at the current left margin, that is, whatever the indention is (0, if no .IN format word is in effect).

Any INDENT format word cancels a current offset and resets the left margin. If you specify a positive or negative increment with the INDENT format word and an offset is in effect, the offset is canceled and the new left margin is computed from the current indent value.

The .IL (INDENT-LINE) format word uses the current margin (the indent value plus the offset value) when computing the margin for the next line.

To achieve a format that has several levels of offsetting, you can combine the .IN and .OF format words.

When you use blank space following the item indicator (for example, the number in a numbered list), HELP may add extra blanks when it justifies the line; if so, the first line may not be aligned with the remainder of the offset item.

Spacing between Lines of Text (.SP Format Word)

If you do not want an input line to be concatenated with the line above it, you must cause a break. To cause a break in a HELP file, begin a line with one or more blank characters (by using the space bar on your terminal keyboard). When HELP reads an input line that begins with a blank character, the formatting process is interrupted; all of the text that has accumulated for the current line is displayed as is, even if more words would have fit on the line; the next input line begins a new output line.

To create paragraphs in text, then, all you have to do is to enter spaces at the beginning of each line that is to begin a new paragraph. For example, the input lines:

```
The quick brown
fox
came over to greet the lazy poodle.
   But the poodle was frightened
and ran away.
```

is displayed by HELP as:

```
The  quick  brown  fox
came over to greet the
lazy poodle.
   But the  poodle was
frightened   and   ran
away.
```

If you want to place blank lines between lines of text, you can press the space bar at least once on a line that has no other text, then press the Return or Enter key.

Instead of entering a blank line, you can use the .SP format word. Thus the input lines:

```
The quick brown fox came over to
greet the lazy poodle.
.sp
But the poodle was frightened
and ran away.
```

are formatted as follows by HELP:

```
The  quick  brown  fox
came over to greet the
lazy poodle.

But  the   poodle  was
frightened   and   ran
away.
```

The .SP format word allows you to enter a numeric parameter indicating how many spaces you want to leave on the text output. For example:

```
.sp 5
```

indicates that you want to leave five lines of space in the text output. You can use multiple spaces when you want a heading or a title to stand out, for example the lines:

```
A Love Story
.sp 3
The quick brown fox
was eager
to meet the pretty poodle.
```

will result in:

```
A Love Story


The  quick  brown  fox
was eager to  meet the
pretty poodle.
```

Translating Output Characters (.TR Format Word)

After HELP has formatted an output line but before it displays that line, HELP may translate any of the characters in that line to a different character representation. You use the .TR format word to request that this translation be done. For example, to request that all blanks (x'40') in the file be displayed as question marks, enter:

```
.tr 40 ?
```

To stop the translation of the question mark as a blank, enter:

```
.tr ? ?
```

Note: When the .TR format word is used without operands, the translation of all characters is stopped.

Appendix A. The CMS Editor

When you issue the EDIT command, the System Product editor automatically places you in CMS editor (EDIT) migration mode. In this mode, you can issue both EDIT and XEDIT subcommands. For complete information on EDIT migration mode, as well as instructions on how to invoke the CMS editor, see the *VM/SP System Product Editor Command and Macro Reference.*

Editing a File

To edit a file means to make changes, additions, or deletions to a CMS file that is on a disk, and to make these changes interactively: you instruct the editor to make a change, the editor does it, and then you request another change.

You can edit a file that does not exist; when you do so, you create the file online, and can modify it as you enter it.

To file a file means to write a file you are editing back onto a disk, incorporating any changes you made during the editing session. When you issue the FILE subcommand to write a file, you are no longer in the environment of the CMS editor, but are returned to the CMS environment. You can, however, write a file to disk and then continue editing it, by using the SAVE subcommand.

An editing session is the period of time during which a file is in your virtual storage area, from the moment you issue the EDIT command and the editor responds EDIT: until you issue the FILE or QUIT subcommands to return to the CMS command environment.

The EDIT Command

When you issue the EDIT command you must specify the filename and filetype of the file you want to edit. If you issue:

```
edit test file
```

CMS searches your A-disk and its extensions for a file with the identification TEST FILE. If the file is not found, CMS assumes that you want to create the file and issues the message:

```
NEW FILE:
EDIT:
```

to inform you that the file does not already exist.

If the file exists on a disk other than your A-disk and its extensions, or if you want to create a file to write on a read/write disk other than your A-disk, you must specify the filemode of the file:

```
edit test file b
```

In this example, your B-disk and its extensions are searched for the file TEST FILE.

After you issue the EDIT command, you are in edit mode, or the environment of the CMS editor. If you have specified the filename and filetype of a file that

already exists, you can now use EDIT subcommands to make changes or corrections to lines in that file. If you want to add records to the file, as you would if you are creating a new file, issue the EDIT subcommand:

```
input
```

to enter input mode. Every line that you enter is considered a data line to be written into the disk file. For most filetypes, the editor translates all of your input data to uppercase characters, regardless of how you enter it. For example, if you create a file and enter input mode as follows:

```
edit myfile test
NEW FILE:
EDIT:
input
INPUT:
This is a file I am
learning to create with the CMS editor.
```

the lines are written into the file as:

```
THIS IS A FILE I AM
LEARNING TO CREATE WITH THE CMS EDITOR.
```

You can use the VM/SP logical line editing symbols to modify data lines as you enter them.

To return to edit mode to modify a file or to terminate the edit session, you must press the Return key on a null line. If you have just entered a data line, for example, and your terminal's typing element or cursor is positioned at the last character you entered, you must press the Return key once to enter the data line, and a second time to enter a null line.

You may also use the logical line end symbol to enter a null line; for example:

```
last line of input#
#
```

Both of these lines cause you to return to edit mode from input mode.

If you do not enter a null line, but enter an EDIT subcommand or CMS command, the command line is written into your file as input. The only exception to this is a line that begins with the characters #CP. These characters indicate that the command is to be passed immediately to CP for processing.

Writing a File Onto Disk

A file you create and the modifications that you make to it during an edit session are not automatically written to a disk file. To save the results, you can do the following:

- Periodically issue the subcommand:

  ```
  save
  ```

  to write onto disk the contents of the file as it exists when you issue the subcommand. Periodically issuing this EDIT subcommand protects your data against a system failure; you can be sure that changes you make are not lost.

- At the beginning of the edit session, issue the AUTOSAVE subcommand, with a number:

  ```
  autosave 10
  ```

  Then, for every tenth change or addition to the file, the editor issues an automatic save request, which writes the file onto disk.

- At the end of the edit session, issue the subcommand:

  ```
  file
  ```

  This subcommand terminates the CMS editor session, writes the file onto disk, replacing a previous file by that name (if one existed), and returns you to the CMS environment.

You can return to the edit environment by issuing the EDIT command, specifying a different file or the same file.

The editor decides which disk to write the file onto according to the following hierarchy:

1. If you specify a filemode on the FILE or SAVE subcommand line, the file is written onto the specified disk.

2. If the current filemode of the file is the mode of a read/write disk, the file is written onto that disk. (If you have not specified a filemode letter, it defaults to your A-disk.)

3. If the filemode is the mode of a read-only extension of a read/write disk, the file is written onto the read/write parent disk.

4. If the filemode is the mode of a read-only disk that is not an extension of a read/write disk, the editor cannot write the file and issues an error message.

See "Changing File Identifiers" for information on how you can tell the editor what disk to use when writing a file.

If you are editing a file and decide, after making several changes, that you do not wish to save the changes, you can use the subcommand:

```
quit
```

No changes that you made since you last used the SAVE subcommand (or the editor last issued an automatic save for you) are retained. If you have just begun an edit session, and have made no changes at all to a file, and for some reason you do not want to edit it at all (for example, you misspelled the name, or want to change a CMS setting before editing the file), you can use the QUIT subcommand instead of the FILE subcommand to terminate the edit session and return to CMS.

A file must have at least one line of data in order to be written.

EDIT Subcommands

While you are in the edit environment, you can issue any EDIT subcommand or macro. An edit macro is an EXEC file that contains a sequence of EDIT

subcommands that execute as a unit. You can create your own EDIT subcommands with the CMS EXEC facility. EDIT subcommands provide a variety of functions. You can:

- Position the current line pointer at a particular line, or record, in a file.
- Control which columns of a file are displayed or searched during an editing session.
- Modify data lines.
- Describe the characteristics that a file and its individual records will have.
- Automatically write and update sequence numbers for fixed-length records.
- Edit files by line number.
- Control the editing session.

Entering EDIT Subcommands

Like CMS commands, EDIT subcommands have a subcommand name and some have operands. In most cases, a subcommand name (or its truncation) can be separated from its operands by one or more blanks, or no blanks. For example, the subcommand lines:

```
type 5
ty     5
t5
```

are equivalent.

Several subcommands also use delimiters, which enclose a character string that you want the editor to operate on. For example, the CHANGE subcommand can be entered:

```
change/apple/pear/
```

The diagonal (/) delimits the character strings APPLE and PEAR. For the subcommands CHANGE, LOCATE, and DSTRING, the first nonblank character following the subcommand name (or its truncation) is considered the delimiter. No blank is required following the subcommand name. In the subcommand:

```
locate $vm/$
```

the dollar sign ($) is the delimiter. You cannot use a / in this case, since the diagonal is part of the character string you want to locate.

When you enter these subcommands, you may omit the final delimiter; for example:

```
dstring/csect
```

You must enter the final delimiter, however, when you specify a global change with the CHANGE subcommand.

For the FIND and OVERLAY subcommands, additional blanks following the subcommand names are interpreted as arguments. The subcommand:

```
find   Pudding
```

requests the editor to locate the line that has " Pudding" in columns 1 through 9. Initial blanks are considered part of the character string.

An asterisk, when used with an EDIT subcommand, may mean "to the end of the file" or "to the record length." For example:

```
delete*
```

deletes all of the lines in a file, beginning with the current line.

```
verify *
```

indicates that the editor should display the entire length of records.

?EDIT:

When you make an error entering an EDIT subcommand, the editor displays the message:

```
?EDIT:  line...
```

where line... is the line, as you entered it, that the editor does not understand.

The Current Line Pointer

When you begin an editing session, a file is copied into virtual storage; in the case of a new file, virtual storage is acquired for the file you are creating. In either case, you can picture the file as a series of records, or lines; these lines are available to you, one at a time, for you to modify or delete. You can also insert new lines or records following any line that is already in the file.

The line that you are currently editing is pointed to by the current line pointer. On a display terminal, this line is highlighted.

What you do during an editing session is:

- Position the current line pointer to access the line you want to edit.
- Edit the line: change character strings in it, delete it or insert new records following it.
- Position the line pointer at the next line you want to edit.

When you are editing a file and you issue an EDIT subcommand that either changes the position of the line pointer or that changes a line, the current line or the changed line (or lines) is displayed. You can also display the current line by using the TYPE subcommand:

```
type
```

If you want to examine more than one line in your file, you can use the TYPE subcommand with a numeric parameter. If you enter:

```
type 10
```

the current line and the nine lines that follow it are displayed; the line pointer then stays positioned at the last line that was displayed.

You can move the line pointer up or down in your file.

UP indicates a location toward the beginning of the file (the first record);

DOWN indicates a location toward the end of the file (the last record).

You use the EDIT subcommands UP and DOWN to move the line pointer up or down one or more lines. For example:

```
up 5
```

moves the current line pointer to a line five lines closer to the beginning of the file, and:

```
down
```

moves the pointer to point at the next sequential record in the file.

You can also request that the line pointer be placed at the beginning, or top of the file, or at the end, or bottom of the file. When you issue the subcommand:

```
top
```

you receive the message:

```
TOF:
```

and the line pointer is positioned at a null line that is always at the top of the file. This null line exists only during your editing session; it is not filed on disk when you end the editing session. When you issue the subcommand:

```
bottom
```

the current line pointer is positioned at the last record in the file. If you now enter input mode, all lines that you enter are appended to the end of the file.

If the current line pointer is at the bottom of the file and you issue the DOWN subcommand, you receive the message:

```
EOF:
```

and the current line pointer is positioned at the end of file, following the last record.

When you are adding records to your file, the current line pointer is always pointing at the line you last entered. When you delete a line from a file, the line pointer moves down to point to the next line down in the file.

Going from edit mode to input mode does not change the current line pointer. If you are creating a new file and you move the current line pointer to make corrections to the lines that you have entered, you must issue the BOTTOM subcommand to begin entering more lines at the end of the file.

The current line pointer is also moved as the result of the LOCATE and FIND subcommands.

FIND is used to get to a line when you know the characters at the beginning of the line.

For example, if you want to change the line:

```
BAXTER      J.F.      065941      ACCNTNT
```

you could first search for it by using the subcommand:

```
find baxter
```

LOCATE is used to get to a line when you do not know the first characters on a line.

You can issue the LOCATE subcommand like this:

```
locate /accntnt/
```

Both of these subcommands work only in a top-to-bottom direction: you cannot use them to position the line pointer above the current line. If you use the FIND or LOCATE subcommands and the target (the character string you seek) is not found, the editor displays a message, and positions the line pointer at the end of the file. Subsequently, if you reissue the subcommand, the editor starts searching at the top of the file.

In a situation like that above, or in a case where you are repetitively entering the same LOCATE or FIND subcommand (if, for example, there are many occurrences of the same character string, but you seek a particular occurrence) you can use the = (REUSE) subcommand. To use the example above, you are looking for a line that contains the string ONCE UPON A TIME, but you do not know that it is above the current line. When you issue the subcommand:

```
locate /once upon a time/
```

the editor does not locate the line, and responds:

```
NOT FOUND
EOF:
```

If you enter:

```
=
```

the editor searches again for the same string, beginning this time at the top of the file, and locates the line:

```
''ONCE UPON A TIME'' IS A COMMON
```

This may still not be the line you are looking for. You can, again, enter:

```
=
```

The LOCATE subcommand is executed again. This time, the editor might locate the line:

```
A STORY THAT STARTED ONCE UPON A TIME
```

Figure A-1 on page A-8 illustrates a simple CMS file, and indicates how the current line pointer would be positioned following a sequence of EDIT subcommands.

Line-Number Editing:

Some fixed-length files are suitable for editing by referencing line numbers instead of character strings. The EDIT subcommands that allow you to change the line pointer position by line number are discussed under "Line-Number Editing."

```
        EDIT PPRINT EXEC
CLP
--->    TOF:
      0 (null line)
      1 &CONTROL OFF
      2 &P =
      3 &IF .&1 EQ . &EXIT 100
      4 &FN = &1
      5 &IF &1 EQ ? &GOTO -TELL
      6 &NFN = &CONCAT $ &1
      7 &IF .&2 EQ . &EXIT 200
      8 &FT = &2
      9 &FM = &3
     10 &IF .&3 NE . &SKIP 2
     11        &FM = A
     12        &SKIP 3
     13 &IF &3 NE ( &SKIP 2
     14        &FM = A
     15        &P = (
     16 &CONTROL ALL
     17 COPY &FN &FT &FM &NFN &FT A ( UNPACK
     18 PRINT &NFN &FT A &P &4 &5 &6 &7 &8 &9 &10 &11 &12 &13 &14
     19 ERASE &NFN &FT A
     20 &EXIT
     21 -TELL     &TYPE THIS EXEC PRINTS A LISTING FROM PACKED FORMAT
        EOF:

The line numbers represented are symbolic: they are not an actual
part of the file, but are used below to indicate at which line the
current line pointer is positioned after execution of the EDIT
subcommand indicated.

    Subcommand          CLP Position
    ----                  ---> 0
    DOWN 5                ---> 5
    UP                    ---> 4
    LOCATE /UNP/          ---> 17
    TYPE 3                ---> 19
    BOTTOM                ---> 21
    DOWN                  ---> EOF:
    FIND -                ---> 21
    TOP                   ---> 0
    CHANGE /EQ/EQ/ 6      ---> 5
    DELETE 2              ---> 7 (lines numbered 5 and 6 are deleted)
    INPUT  *              ---> the line just entered (between 7 and 8)
```

Figure A-1. Positioning the Current Line Pointer

Verification and Search Columns

There are two EDIT subcommands you can use to control what you and the editor "see" in a file.

VERIFY controls what you see displayed.

ZONE controls what columns the editor searches.

Normally, when you edit a file, every request that you make of the editor results in the display of one or more lines at your terminal. If you do not want to see the lines, you can specify:

```
verify off
```

Alternatively, if you want to see only particular columns in a file, you can specify the columns you wish to have displayed:

```
verify 1 30
```

Some filetypes have default values set for verification, which usually include those columns in the file that contain text or data, and exclude columns that contain sequence numbers. If a verification column is less than the record length, you can specify:

```
verify *
```

to indicate that you want to see all columns displayed.

In conjunction with the VERIFY subcommand, you can use the ZONE subcommand to tell the editor within which columns it can search or modify data. When you issue the subcommand:

```
zone 20 30
```

The editor ignores all text in columns 1-19 and 31 to the end of the record when it searches lines for LOCATE, CHANGE, ALTER, and FIND subcommands. You cannot unintentionally modify data outside of these fields; you must change the zones in order to operate on any other data.

The zone setting also controls the truncation column for records when you are using the CHANGE subcommand; for more details, see "Setting Truncation Limits."

Changing, Deleting, and Adding Lines

You can change character strings in individual lines of data with the CHANGE subcommand. A character string may be any length, or it may be a null string. Any of the characters on your terminal keyboard, including blanks, are valid characters. The following example shows a simple data line and the cumulative effect of CHANGE subcommands.

```
ABC ABC ABC
```

is the initial data line.

```
CHANGE /ABC/XYZ/
```

changes the first occurrence of the character string "ABC" to the string "XYZ."

```
XYZ ABC ABC
```

```
CHANGE /ABC//
```

deletes the character string "ABC" and concatenates the characters on each side of it.

```
XYZ  ABC
```

```
CHANGE //ABC/
```

inserts the string "ABC" at the beginning of the line.

```
ABCXYZ  ABC
```

```
CHANGE /XYZ /XYZ/
```

deletes one blank character following "XYZ."

```
ABCXYZ ABC
```

```
CHANGE /C/C /
```

adds a blank following the first occurrence of the character "C."

```
ABC XYZ ABC
```

is the final line.

The ALTER Subcommand:

You can use the ALTER subcommand to change a single character; the ALTER subcommand allows you to specify a hexadecimal value so that you can include characters in your files for which there are no keyboard equivalents. Once in your file, these characters appear during editing as nonprintable blanks. For example, if you input the line:

```
IF A = B THEN
```

in edit mode and then issue the subcommand:

```
alter = 8c
```

the line is displayed:

```
IF A   B THEN
```

If you subsequently print the file containing this line on a printer equipped to handle special characters, the line appears as:

```
IF A ≤ B THEN
```

since X'8C' is the hexadecimal value of the special character.

Either or both of the operands on the ALTER subcommand can be hexadecimal or character values. To change the X'8C' to another character, for example <, you could issue either:

```
alter 8c 4c
-- or  --
alter 8c <
```

The OVERLAY Subcommand:

The OVERLAY subcommand allows you to replace characters in a line by spacing the terminal's typing element or cursor to a particular character position to make character-for-character replacements, or overlays. For example, given the line:

```
ABCDEF
```

the subcommand:

```
overlay xyz
```

results in the line:

```
XYZDEF
```

A blank entered on an OVERLAY line indicates that the corresponding character is not to be changed; to replace a character with a blank, use an underscore character (_). Given the above line, XYZDEF, the subcommand:

```
overlay ___  3
```

results in:

```
   DE3
```

(The "D" is preceded by blanks in columns 1, 2, and 3.)

Global Changes

You can make global or repetitive changes with the CHANGE and ALTER subcommands. On these subcommand lines, you can include operands that indicate:

- The number of lines to be searched for a character or character string. An asterisk (*) indicates that all lines, from the current line to the end of the file, are to be searched.
- Whether only the first occurrence or all occurrences on each line are to be modified. An asterisk (*) indicates all occurrences. If you do not specify an asterisk, only the first occurrence on any line is changed.

For example, if you are creating a file that uses the (•) special character (X'AF') and you do not want to use the ALTER subcommand each time you need to enter the •; you could use the character ¬ as a substitute each time you need to enter a •. When you are finished entering input, move the current line pointer to the top of the file, and issue the global ALTER subcommand:

```
top#alter ¬ af * *
```

All occurrences of the character ¬ are changed to X'AF'. The current line pointer is positioned at the end of the file.

When you use a global CHANGE subcommand, you must be sure to use the final delimiter on the subcommand line. For example:

```
change /hannible/hannibal/ 5
```

This subcommand changes the first occurrence of the string "HANNIBLE" on the current line and the four lines immediately following it.

You can also make global changes with the OVERLAY subcommand, by issuing a REPEAT subcommand just prior to the OVERLAY subcommand. Use the REPEAT subcommand to indicate how many lines you want to be affected. For example, if you are editing a file containing the three lines:

```
A
B
C
```

with the current line pointer at line "A," issuing the subcommands:

```
repeat 3
overlay   |   |   |
```

results in:

```
A |   |   |
B |   |   |
C |   |   |
```

The current line pointer is now positioned at the line beginning with the character "C."

Deleting Lines

You delete lines from a file with the DELETE subcommand; to delete more than one line, specify the number of lines:

```
delete 6
```

Or, if you want to delete all the lines from the current line to the end of the file, use an asterisk (*):

```
delete *
```

If you want to delete an undetermined number of lines, up to a particular character string, you can use the DSTRING subcommand:

```
dstring /weather/
```

When this subcommand is entered, all the lines from and including the current line down to and including the line just above the line containing the character string "WEATHER" are deleted. The current line pointer is positioned at the line that has "WEATHER" on it.

If you want to replace a line with another line, you can use the REPLACE subcommand:

```
replace *******
```

The current line is deleted and the line "*******" is inserted in its place. The current line pointer is not moved.

To replace an existing line with many new lines, you can issue the REPLACE subcommand with no new data line:

```
replace
```

The editor deletes the current line and enters input mode.

Inserting Lines

You can insert a single line of data between existing lines using the INPUT subcommand followed by the line of data you want inserted. For example:

```
input * this subroutine is for testing only
```

inserts a single line following the current line. If you want to insert many lines, you can issue the INPUT subcommand to enter input mode.

You can also add new lines to a file by using the GETFILE subcommand. This allows you to copy lines from other files to include in the file you are editing or creating. For example:

```
getfile single items c
```

inserts all the lines in the file SINGLE ITEMS C immediately following the current line pointer. The line pointer is positioned at the last line that was read in. You could also specify:

```
getfile double items c 10 25
```

to copy 25 lines, beginning with the tenth line, from the file DOUBLE ITEMS C.

The $MOVE and $DUP EDIT macros provide two additional ways of adding lines into a file in a particular position. The $MOVE macro moves lines from one place in a file to another, and deletes them from their former position. For example, if you want to move 10 lines, beginning with the current line, to follow a line 9 lines above the current line, you can enter:

```
$move 10 up 8
```

The $DUP macro duplicates the current line a specified number of times, and inserts the new lines immediately following the current line. For example:

```
$dup 3
```

creates 3 copies of the current line, and leaves the current line pointer positioned at the last copy.

Describing Data File Characteristics

When you issue the EDIT command to create a new file, the editor checks the filetype. If it is one of the reserved filetypes, the editor may assign particular attributes to it, which can simplify the editing process for you. The default attributes assigned to most filetypes are as follows:

- Fixed-length, 80-character records
- All alphabetic characters are translated to uppercase, regardless of how they are entered
- Input lines are truncated in column 80

- Tab settings are in columns 1, 6, 11, 16, 21, ... 51, 61, and so on, and the tab characters are expanded to blanks
- Records are not serialized

The filetypes for some CMS commands and for the language processors deviate from these default values. Some of the attributes assigned to files and how you can adjust them to suit your needs are discussed below.

Record Length

You can specify the logical record length of a file you are creating on the EDIT command line:

```
edit new file (lrecl 130
```

If you do not specify a record length, the editor assumes the following defaults:

- For editing old files, the existing record length is used.
- For creating new files, the following default values are in effect:

Filetype	Record Length	Format
EXEC	80 characters	Variable
FREEFORT	81 characters	Variable
LISTING	121 characters	Variable
SCRIPT	132 characters	Variable
VSBDATA	132 characters	Variable
All others	80	Fixed

If you edit a variable-length file and the existing record length is less than the default for the filetype, the record length is taken from the default value.

When you use the LRECL option of the EDIT command you can override these default record lengths; you can also change the record lengths of existing files to make them larger, but not smaller.

If you try to override the record length of an existing file and make it smaller, the editor displays an error message, and you must issue the EDIT command again with a larger record length. For example, suppose you have on your B-disk a file named MYFILE FREEFORT, which was created with the default record length of 81. If you try to edit that file by issuing:

```
edit myfile freefort b (lrecl 72
```

the editor displays the message:

```
GIVE A LARGER RECORD LENGTH.
```

You must then issue the EDIT command again and either specify a length of 81 or more, or allow it to default to the current record length of the file.

You can use the COPYFILE command to increase or decrease the record length of a file before you edit it. For example, if you have fixed-length, 132-character records in a file, and you want to truncate all the records at column 80 and create a file with 80-character records, you could issue the command:

```
copyfile extra funds a (lrecl 80
```

Long Records

The largest record you can edit with the editor is 160 characters. A file with record length up to 160 bytes (for example, a listing file created by a DOS program) can be displayed and edited.

The largest record you can create with the CMS editor, however, is 130 characters using a 3270 display terminal and 134 characters using a typewriter terminal such as a 2741 or 1050. If you enter more than 130 characters on a 3270, the record is truncated to 130 characters when you press the Enter key.

> **Note:** As the line is truncated to 130 characters, the CMS editor will not know the actual line length entered, and will not issue the "TRUNCATED" message. If you type more than 134 characters on a line using a typewriter terminal, CP generates an attention interruption to your virtual machine and the input line is lost when you press the Return Key.

For most purposes, you will not need to create records longer than 130 characters. If it is necessary, you can expand a record that you have entered. You do this by issuing the CHANGE subcommand with operands, to add more characters to the record (for example, by changing a 1-character string to a 31-character string). However, if a record is longer than 130 characters, the CHANGE subcommand without operands will cause truncation to 130 characters.

You cannot create a record that is longer than the record length of the file. For example, if the file you are editing has a default record length of 80, or if you specified LRECL 80 when you created the file, the editor truncates all records to 80 characters.

Record Length and File Size

There is a relationship between the record length of a file and the maximum number of records it can contain. Figure A-2 shows the approximate number of records, rounded to the nearest hundred, that the CMS editor can handle in a virtual machine with different amounts of virtual storage.

Record Length	Virtual Machine Size 320K	512K	768K	1024K
80 Characters	1700	3800	6800	9800
120 Characters	1100	2600	4700	6800
132 Characters	1100	2400	4300	6200
160 Characters	900	2000	3600	5100

Figure A-2. Number of Records Handled by the CMS Editor

Record Format

With the CMS editor, you can create either fixed- or variable-length files. Except for the filetypes EXEC, LISTING, FREEFORT, SCRIPT, and VSBDATA, all the files you create have fixed-length records, by default. You can change the format of a file at any time during an editing session by using the RECFM subcommand:

```
recfm v
```

This changes the record format to variable-length. This does not change the record length; in order to add new records with a greater length, you must write the file onto disk and then reissue the EDIT command using the LRECL option.

The COPYFILE command also has an RECFM option, so that you can change the record format of a file without editing it. The command:

```
copyfile * requests a1 (recfm v trunc
```

changes the record formats of all the files with a filetype of REQUESTS on your A-disk to variable-length. The TRUNC option specifies that you want trailing blanks removed from each of the records. When you are editing a file with variable-length records, trailing blanks are truncated when you write the file onto disk with the FILE or SAVE subcommand. (In VSBDATA files, however, blanks are not truncated.)

Using Special Characters

The IMAGE and CASE subcommands control how data, once entered on an input line, is going to be represented in a file. The specific characters affected, and the subcommands that control their representation, are:

- Alphabetic characters: CASE subcommand
- Tab characters (X'05'): IMAGE subcommand (ON and OFF operands)
- Backspaces (X'16'): IMAGE subcommand (CANON operand)

Alphabetic Characters

If you are using a terminal that has only uppercase characters, you do not need to use the CASE subcommand; all of the alphabetic characters you enter are uppercase. On terminals equipped with both uppercase and lowercase letters, all lowercase alphabetic characters are converted to uppercase in your file, regardless of how you enter them. If you are creating a file and you want it to contain both uppercase and lowercase letters you can use the subcommand:

```
case m
```

The "M" stands for "mixed." This attribute is not stored with the file on disk. If you create a new file, and you issue the CASE M subcommand, all the lowercase characters you enter remain in lowercase. If you subsequently file the file and later edit it again, you must issue the CASE M subcommand again to locate or enter lowercase data.

There are two reserved filetypes for which uppercase and lowercase is the default. These are SCRIPT and MEMO, both of which are text or document-oriented filetypes. For most programming applications, you do not need to use lowercase letters.

Tab Characters

Logical tab settings indicate the column positions where fields within a record begin. These logical tab settings do not necessarily correspond to the physical tab settings on a typewriter terminal. What happens when you press the Tab key on a typewriter terminal depends on whether the image setting is on :i1,image setting, effect on tab characters or off. The default for all filetypes except SCRIPT is IMAGE ON. You can change the default by issuing the subcommand:

```
image off
```

If the image setting is on, when you press the Tab key the editor replaces the tab characters with blanks, starting at the column where you pressed the Tab key, and ending at the last column before the next logical tab setting. The next character entered after the tab becomes the first character of the next field. For example, if you enter:

```
tabset 1 15
```

and then enter a line that begins with a tab character, the first data character following the tab is written into the file in column 15, regardless of the tab stop on your terminal.

If the image setting is off, the tab character, X'05', is inserted in the record, just as any other data character is inserted. No blanks are inserted.

If you want to insert a tab character (X'05') into a record and the image setting is on, you can do one of the following:

1. Set IMAGE OFF before you enter or edit the record, and then use the Tab key as a character key.

2. Enter some other character at the appropriate place in the record, and use the ALTER subcommand to alter that character to a X'05'.

Setting Tabs: When you create a file, there are logical tab settings in effect, so that you do not need to set them. The default values for the language processors correspond to the columns used by those processors. If you want to change them, or if you are creating a file with a nonreserved filetype, you may want to set them yourself. Use the TABSET subcommand, for example:

```
tabset 1 12 20 28 72
```

Then, regardless of what physical tab stops are in effect for your terminal, when you press the Tab key with image setting ON, the data you enter is spaced to the appropriate columns.

See Figure A-3 for the default tab settings used by the CMS editor.

Filetype	Default Tab Settings
ASSEMBLE, MACRO, COPY, UPDATE, UPDT, ASM3705, MACLIB, XEDIT	1, 10, 16, 30, 35, 40, 45, 50, 55, 60, 65, 70
AMSERV, ESERV	2, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60
FORTRAN	1, 7, 10, 15, 20, 25, 30, 80
FREEPORT	9, 15, 18, 23, 33, 38, 81
DIRECT, JOB	1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75
EXEC, CNTRL	1, 5, 8, 17, 27, 31
COBOL	1, 8, 12, 20, 28, 36, 44, 68, 72, 80
BASIC, BASDATA, VSBASIC	7, 10, 15, 20, 25, 30, 80

Figure A-3. Default Tab Settings

Filetype	Default Tab Settings
VSBDATA, SCRIPT, MEMO, LISTING, *******	1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 105, 110, 115, 120
PLI, PLIOPT	2, 4, 7, 10, 13, 16, 19, 22, 25, 31, 37, 43, 49, 55, 79, 80

Figure A-3. Default Tab Settings

When you are specifying tab settings for files, the first tab setting you specify should be the column in which you want your data to begin. The editor will not allow you to place data in a column preceding this one. For example, if you issue:

```
tabset 5 10 15 20
```

and enter an input line:

```
input This is a line
```

Columns 1, 2, 3, and 4 contain blanks; text begins in column 5.

Backspaces

For most of your applications, you do not need to underscore or overstrike characters or character strings. If you are using a typewriter terminal and are typing files that use backspaces and underscores, you should use either the IMAGE OFF or IMAGE CANON subcommands so that the editor handles the backspaces properly. IMAGE CANON is the default value for SCRIPT files.

CANON means that regardless of how the characters are keyed in (characters, backspaces, underscores), the editor orders, or canonizes, the characters in the file as: character-backspace-underscore, character-backspace-underscore, and so on. If, for example, you want an input line to look like:

```
ABC
```

You could enter it as:

```
ABC, 3 backspaces, 3 underscores
          - or -
3 underscores, 3 backspaces, ABC
```

A typewriter types out the line in the following order:

```
A backspace, underscore
B backspace, underscore
C backspace, underscore, which results in:
ABC
```

If you need to modify a line that has backspaces, and you do not want to rekey all of the characters, backspaces, and overstrike characters in a CHANGE or REPLACE subcommand, you can use the ALTER subcommand to alter all of the backspaces to some other character and use a global CHANGE command. For example, the following sequences shows how to delete all of the backspace characters on a line:

```
AAAAA
alter 16 + 1 *
_+A_+A_+A_+A_+A
change /_+// 1 *
AAAAA
```

This technique may also be useful on a display terminal.

Setting Truncation Limits

Every CMS file that you edit has a truncation column setting: this column represents the last character position in a record into which you can enter data. When you try to input a record that is longer than the truncation column, the record is truncated, and the editor sends you a message telling you that it has been truncated.

You can change the truncation column setting with the TRUNC subcommand. For example, if you are creating a file with a record length of 80 and wish to insert some records that do not extend beyond column 20, you could issue the subcommand:

```
trunc 20
```

Then, when you enter data lines, any line that is longer than 20 characters is truncated and the editor sends you a message. If you are entering data in input mode, your virtual machine remains in input mode.

When you use the CHANGE subcommand to modify records, the column at which truncation occurs is determined by the current zone setting. If you change a character string in a line to a longer string, and the resultant line extends beyond the current end zone, you receive the message:

```
TRUNCATED.
```

If you need to create a line longer than the current end zone setting, use the ZONE subcommand to increase the setting. The subcommand:

```
zone  1 *
```

extends the zone to the record length of the file. If the end zone already equals the record length, you have to write the file onto disk and reissue the EDIT subcommand specifying a longer record length.

For most filetypes, the truncation and end zone columns are the same as the record length. For some filetypes, however, data is truncated short of the record length. The default truncation and end zone columns are:

Column	Filetype
71	ASSEMBLE, MACRO, UPDATE, UPDTxxxx
72	AMSERV, COBOL, DIRECT, FORTRAN, PLI, PLIOPT

All other filetypes are truncated at their record length.

You can, when creating files for your own uses, set truncation columns so that data does not extend beyond particular columns.

Entering a Continuation Character in Column 72

When you are using the editor to enter source records for an assembler language program and you need to enter a continuation character in column 72, or whenever you want to enter data outside a particular truncation setting, you can use the following technique.

Note: This technique will not work if CANON is specified on the IMAGE subcommand.

1. Change the truncation setting to 72, so that the editor does not truncate the continuation character:

   ```
   trunc 72
   ```

2. Use the TABSET subcommand to set the left margin at column 72:

   ```
   tabset 72
   ```

3. Use the OVERLAY subcommand to overlay an asterisk in column 72:

   ```
   overlay *
   ```

   Since the left margin is set at 72, the OVERLAY subcommand line results in the character * being placed in column 72.

4. Restore the editor truncation and tab settings:

   ```
   trunc 71
   tabset 1 10 16 31 36 41 51 61 71 81
   ```

Note: If you issue the PRESERVE subcommand before you change the truncation and tab settings, then after you enter the OVERLAY subcommand, you can restore them with the RESTORE subcommand. See "Preserving and Restoring CMS Editor Settings."

Use the $MARK Edit Macro: Another way to insert a continuation character is to use the $MARK edit macro. You can find out if the $MARK edit macro is available on your system by entering, in the CMS or CMS subset environment:

```
listfile $mark exec *
```

If it is not available on your system, you can create the $MARK edit macro for your own use. See "Writing Edit Macros" in Appendix B.

If you have the $MARK macro, then when you need to enter a continuation character, you can enter a null line to get into edit mode, issue the command:

```
$mark
```

and then return to input mode to continue entering text.

Serializing Records

Some CMS files that you create are automatically serialized for you. This means that columns 73 to 80 of each record contain an identifier in the form:

```
cccxxxxx
```

where ccc are the first three characters of the filename and xxxxx is a sequence number. Sequence numbers begin at 00010 and are incremented by 10.

The filetypes that are automatically serialized in columns 73 to 80 are:

```
ASSEMBLE
DIRECT
MACRO
FORTRAN
COBOL
PLI
PLIOPT
UPDATE
UPDTxxxx
```

You can serialize any file that has fixed-length, 80-character records by using the SERIAL subcommand:

```
serial on
```

The SERIAL subcommand can also be used to:

- Assign a particular three-character identifier:

  ```
  serial abc
  ```

- Specify that all eight bytes of the sequence field be used to contain numbers:

  ```
  serial all
  ```

- Specify a sequence increment other than 10:

  ```
  serial on 100
  -- or --
  serial ccc 100
  ```

- Indicate that no sequence numbers are to be assigned to new records being inserted:

  ```
  serial off
  ```

When you create a file or edit a file with sequence numbers, the sequence numbers are not written or updated until you issue a FILE or SAVE subcommand. Because the end verification columns for the filetypes that are automatically serialized are the same as their truncation columns, you do not see the serial numbers unless you specify:

```
verify *
-- or --
verify 80
```

Although the serial numbers are not displayed while you edit the file, they do appear on your output listings or printer files.

If you are editing files with the following filetypes:

```
BASIC
VSBASIC
FREEFORT
```

the sequence numbers are on the left. For BASIC and VSBASIC files, columns 1-5 are used; numbers are blank-padded to the left. For FREEFORT files, the sequence numbers use columns 1-8, and are zero-padded to the left. To edit these files, you should use line-number editing, which is discussed next.

Line-Number Editing

To edit a file by line numbers means that when you are adding new lines to a file or referencing lines that you wish to change, you refer to them by their line, or sequence numbers, rather than by character strings. You can use right line-number editing only on files with fixed-length, 80-character records.

If you want to edit by line numbers, issue the subcommand:

```
linemode right
 -- or --
linemode left
```

where:

right indicates that the sequence numbers are on the right, in columns 76-80

and

left indicates you want sequence numbers on the left in columns 1-5.

LINEMODE LEFT is the default for BASIC, VSBASIC, and FREEFORT files. You do not have to specify it. You must specify LINEMODE for files with other filetypes.

If you specify LINEMODE RIGHT to use line-number editing on a typewriter terminal, the line numbers are displayed on the left, as a convenience, while you edit the file.

When you are using line-number editing in input mode, you are prompted to enter lines; the line numbers are in increments of 10. For example, when you are creating a new file, you are prompted for the first line number as follows:

```
10
```

On a typewriter terminal, you enter your input line following the 10. When you press the carriage return, you are prompted again:

```
20
```

and you continue entering lines in this manner until you enter a null line.

You can change the prompting increment to a larger or smaller number with the PROMPT subcommand:

```
prompt 100
```

When you are in edit mode you can locate a line by giving its line number:

```
700
```

This is the nnnnn subcommand. In line-number editing, you use it instead of the INPUT subcommand to insert a single line of text. For example:

```
905 x = a * b
```

inserts the text line "X = A * B" in the proper sequence in the file. If you use "nnnnn text" specifying the number of a line that already exists, that line is replaced; the current line pointer is moved to point to it.

The EDIT subcommands that you normally use for context editing, such as CHANGE, ALTER, LOCATE, UP, DOWN, and so forth, can also be used when you are line-number editing; their operation does not change.

Renumbering Lines

When you are using line-number editing, the editor uses the prompting increment set by the PROMPT subcommand. However, when you begin adding lines of data between existing lines, the editor uses an algorithm to select a line number between the current line number and the next line number. If a prompting number cannot be generated because the current line number and the next line number differ only by one, the editor displays the message:

```
RENUMBER LINES
```

and you must resequence the line numbers in the file before you can continue line-number editing.

You can resequence the line numbers in one of three ways:

1. If you are a VSBASIC or FREEFORT user, you may use the RENUM subcommand:

   ```
   renum
   ```

   This subcommand resolves all references to lines that are renumbered.

2. If you are using right-handed line-number editing, you must:

   a. Turn off line-number editing:

      ```
      linemode off
      ```

   b. If you want to change the three-character identifier or specify eight-character sequence numbers, issue the SERIAL subcommand, for example:

      ```
      serial all
      ```

      If you want to use the default serialization setting, you do not need to issue the SERIAL subcommand.

   c. Issue the SAVE subcommand:

      ```
      save
      ```

   d. Reissue the LINEMODE subcommand and continue line-number editing:

      ```
      linemode right
      ```

3. If you are using left-handed line-number editing for a filetype other than VSBASIC or FREEFORT, you must manually change individual line numbers using EDIT subcommands. In order to modify the line numbers, you must change the zone setting and the tab setting:

   ```
   zone 1 *
   tabset 1 6
   ```

   so that you can place data in columns 1 through 6.

When you are using right-handed line-number editing, and a FILE, SAVE, or automatic save request is issued, the editor does not resequence the serial numbers, but displays the message:

```
RESERIALIZATION SUPPRESSED
```

so that the lines numbers that are currently saved on disk match the line numbers in the file. You must cancel line-number editing (using the LINEMODE OFF subcommand) before you can issue a FILE or SAVE subcommand if you want to update the sequence numbers.

Controlling the CMS Editor

There are a number of EDIT subcommands that you can use to maximize the use of the editor in CMS. A few techniques are suggested here; as you become more familiar with VM/SP and CMS you will develop additional techniques for your own applications.

Communicating with CMS and CP

Often during a terminal session, you may need to issue a CMS command or a CP command. You can issue certain CMS commands and most CP commands without terminating the edit session. The EDIT subcommand CMS places your virtual machine in the CMS subset mode of the editor, where you can issue CMS commands that do not modify your virtual storage. Remember that the editor is using your virtual storage; if you overlay it with any other command or program, you will not be able to finish your editing.

One occasion when you may want to enter CMS subset is when you want to issue a GETFILE subcommand for a file on one of your virtual disks and you have not accessed the disk. You can enter:

```
cms
```

The editor responds:

```
CMS SUBSET
```

Then you can enter:

```
access 193 b/a
return
get setup script b
```

The special CMS SUBSET command RETURN returns your virtual machine to edit mode.

You can enter CP commands from CMS subset, or you can issue them directly from edit mode or input mode with the #CP function. For example, if you are inputting lines into a file and another user sends you a message, you can reply without leaving input mode:

```
#cp m opr i will call you later
```

If you enter #CP without specifying a command line, you receive the message:

```
CP
```

which indicates that your virtual machine is in the CP command environment, and you can issue CP commands. You would not, however, want to issue any CP command that would modify your virtual storage or alter the status of the disk on which you want to write the file.

To return to edit or input mode from CP, use the CP command, BEGIN. If you are working at a display terminal and the screen image does not reappear, enter the TYPE command to cause the editor to redisplay the screen.

Changing File Identifiers

There are several methods you can use to change a file identifier before writing the file onto disk. You can use the FNAME and FMODE subcommands to change the filename or filemode, or you can issue a FILE or SAVE subcommand specifying a new file identifier.

For example, if you want to create several copies of a file while you are using the editor, you can issue a series of FNAME subcommands, followed by SAVE subcommands, as follows:

```
edit test file
EDIT:
.
.
.
fn test1#save
.
.
.
fn test2#save
.
.
.
fn test3#file
```

Or, you could issue the SAVE and FILE subcommands as follows:

```
edit test file
.
.
.
save test1
.
.
.
save test2
.
.
.
file test3
```

In both of the preceding examples, when the FILE subcommand is executed, there are files named TEST FILE, TEST1 FILE, TEST2 FILE, and TEST3 FILE. The original TEST FILE is unchanged.

To change the filemode letter of a disk, use the FMODE subcommand. You can do this in cases where you have begun editing a file that is on a read-only disk, and want to write it. Since you cannot write a file onto a read-only disk, you can issue the FMODE subcommand to change the mode before filing it:

```
fmode a
file
```

Or, you can use the FILE (or SAVE) subcommand specifying a complete file identifier:

```
file test file a
```

You should remember, however, that when you write a file onto disk, it replaces any existing file that has the same identifier. The editor does not issue any warning or informational messages. If you are changing a file identifier while you are editing the file, you must be careful that you do not unintentionally overlay existing files. To verify the existence of a file, you can enter CMS subset and issue the STATE or LISTFILE commands.

Controlling the CMS Editor's Displays

When you are using a typewriter terminal, you may not always want to see the editor verify the results of each of your subcommands. Particularly when you are making global changes, you may not want to see each line displayed as it is changed. You can issue the VERIFY subcommand with the OFF operand to instruct the editor not to display anything unless specifically requested. After you issue:

```
verify off
```

lines that are normally displayed as a result of a subcommand that moves the current line pointer (UP, DOWN, TOP, BOTTOM, and so forth), or that changes a line (CHANGE, ALTER, and so forth), are not displayed. If the current line pointer moves to the end of the file, however, the editor always displays the EOF: message.

If you are editing with verification off, then you must be particularly careful to stay aware of the position of your current line pointer. You can display the current line at any time using the TYPE subcommand:

```
type
```

Long and Short Error Messages: When you enter an invalid subcommand while you are using the editor, the editor normally responds with the error message:

```
?EDIT: line...
```

displaying the line that it did not recognize. If you prefer, you can issue the SHORT subcommand so that instead of receiving the long form of the error, you receive the short form, which is:

```
¬
```

When you issue an invalid edit macro request (any line that begins with a $), you receive the message:

```
¬$
```

To resume receiving the long form of the error message, use the LONG subcommand:

```
long
```

LONG and SHORT control the display of the error message regardless of whether you are editing with verification on or off.

On a display terminal, all EDIT messages that are displayed at the top of the screen, including error messages and "?EDIT:" messages, are highlighted.

Preserving and Restoring CMS Editor Settings

The PRESERVE and RESTORE subcommands are used together; the PRESERVE subcommand saves the settings of the EDIT subcommands that control the file format, message and verification display, and file identifier. If you are editing a file and you want to temporarily change some of these settings, issue the PRESERVE subcommand to save their current status. When you have finished your temporary edit project, issue the RESTORE subcommand to restore the settings.

For example, if you are editing a SCRIPT file and want to change the image setting to create a particular format, you can enter:

```
preserve
image on
tabset 1 15 40 60 72
zone 1 72
trunc 72
```

When you have finished entering data using these settings, you can issue the subcommand:

```
restore
```

to restore the default settings for SCRIPT filetypes.

X, Y, =, ? Subcommands

The X, Y, =, and ? subcommands all perform very simple functions that can help you to extend the language of the CMS editor. They allow you to manipulate, reuse, or interrogate EDIT subcommands.

If you have an editing project in which you have to execute the same subcommand a number of times, you can assign it to the X or Y subcommands, as follows:

```
x locate /insert here/
y getfile insert file c
```

Each time that you enter the X subcommand:

```
x
```

the command line LOCATE /INSERT HERE/ is executed, and every time you enter the Y subcommand:

```
y
```

the GETFILE subcommand is executed.

When you specify a number following an X or Y subcommand, the subcommand assigned to X or Y is executed the specified number of times; for example:

```
x locate /aa/
x 10
```

the LOCATE subcommand line is executed 10 times before you can enter another EDIT subcommand.

Another method of re-executing a particular subcommand is to use the = (REUSE) subcommand. For example, if you enter:

```
locate /ard/
AARDVARK
=======
```

the LOCATE subcommand is re-executed seven times.

What the = (REUSE) subcommand actually does is to stack the subcommand in the console stack. Since CMS, and the editor, read from the console stack before reading from the terminal, the lines in the stack execute before a read request is presented to the terminal. When you enter multiple equal signs, the subcommand is stacked once for each equal sign you enter.

You can also stack an additional EDIT subcommand following an equal sign. The subcommand line is also stacked, but it is stacked LIFO (last-in, first-out) so that it executes before the stacked subcommand. For example, if you enter:

```
delete
= next
```

a DELETE subcommand is executed, then a DELETE subcommand is stacked, and a NEXT subcommand is stacked in front of it. Then the stacked lines are read in and executed. The above sequence has the same effect as if you enter:

```
delete
next
delete
```

In addition to stacking the last subcommand executed, you can also find out what it was, using the ? subcommand. For example, if you enter:

```
next 10
?
```

the editor displays:

```
NEXT 10
```

Since the subcommand line NEXT 10 was the last subcommand entered, if you enter an = subcommand, it is executed again. You cannot stack a ? subcommand.

> **Note:** The ? subcommand, on a display terminal, copies the last EDIT subcommand into the user input area, where you may modify it before re-entering it.

What To Do When You Run Out of Space

There are two situations that may prevent you from continuing an edit session or from writing a file onto disk. You should be aware of these situations, know how to avoid them, and how to recover from them, should they occur.

When you issue the EDIT command to edit a file, the editor copies the file into virtual storage. If it is a large file, or you have made many additions to it, the editor may run out of storage space. If it does, it issues the message:

```
AVAILABLE STORAGE IS NOW FULL
```

When this happens, you cannot make any changes or additions to the file unless you first delete some lines. If you attempt to add a line, the editor issues the message:

```
NO ROOM
```

If you were entering data in input mode, your virtual machine is returned to edit mode, and you may receive the message:

```
STACKED LINES CLEARED
```

which indicates that any additional lines you entered are cleared and will not be processed.

You should use the FILE subcommand to write the file onto disk. If you want to continue editing, you should see that the editor has more storage space to work with. To do this, you can find out how large your virtual machine is and then increase its size. To find out the size, issue the CP QUERY command:

```
cp query virtual storage
```

If the response is:

```
STORAGE =  256K
```

You might want to redefine your storage to 512K. Use the CP command DEFINE, as follows:

```
 cp define storage 512k
```

This command resets your virtual machine, and you must issue the CP IPL command to reload the CMS system before you can continue editing.

If a file is very large, the editor may not have enough space to allow you to edit it using the EDIT command. The message:

```
DMSEDI132S FILE 'fn ft fm' TOO LARGE
```

indicates that you must obtain more storage space before you can edit the file. If this is the case, or if you are editing large files, you should redefine your storage before beginning the terminal session. If this happens consistently, you should see your installation support personnel about having the directory entry for your userid updated so that you have a large storage size to begin with.

Splitting CMS Files Into Smaller Files

If the file you are editing is too large, and the data it contains does not have to be in one file, you can split the file into smaller files, so that it is easier to work with. Two of the methods you can use to do this are described below.

Use the COPYFILE Command: You can use the COPYFILE command to copy portions of a file into separate files, and then delete the copied lines from the original file. For example, if you have a file named TEST FILE that has 1000 records, and you want to split it into four files, you could enter:

```
copyfile test file a test1 file a (from 1 for 250
copyfile test file a test2 file a (from 251 for 250
copyfile test file a test3 file a (from 501 for 250
copyfile test file a test4 file a (from 751 for 250
```

When these COPYFILE commands are complete, you have four files containing the information from the original TEST FILE, which you can erase:

```
erase test file
```

Use the Editor: If you use the editor to create smaller files, you can edit them as you copy them, that is, if you have other changes that you want to make to the data. To copy files with the editor, you use the GETFILE subcommand. Using the file TEST FILE as an example, you might enter:

```
edit test1 file
getfile test file a 1 250
.
.
.
file
edit test2 file
getfile test file a 251 250
.
.
.
```

Again, you could erase the original TEST FILE when you are through with your edit session.

When Your Disk Is Full

When you enter a FILE or SAVE subcommand or when an automatic save request is issued, the editor writes a copy of the file you are editing onto disk, and names it EDIT CMSUT1. If this causes the disk to become full, you receive the message:

```
DMSBWR170S DISK 'mode(cuu)' IS FULL
```

The editor erases the workfile, and issues the message:

```
SET NEW FILEMODE, OR ENTER CMS SUBSET AND CLEAR SOME SPACE
```

The original file (as last written onto disk) remains unchanged. You can use the CMS subcommand to enter CMS subset, and erase any files that you do not need. You can use the LISTFILE command to list the files on the disk, then the ERASE command to erase the unwanted files.

If you cannot erase any of the files on the disk, there are several alternate recovery paths you can take:

1. If you have another read/write disk accessed, you can use the FMODE subcommand to change the filemode of the file, so that when you file it, it is written to the other disk. If you have a read/write disk that is not accessed, you can access it in CMS subset. After filing the file on the second disk, erase the original copy, and then use the COPYFILE command to transfer the file back to its original disk.

2. If you do not have any other read/write disk in your virtual machine, you may be able to transfer some of your files to another user, using either the SENDFILE, PUNCH or DISK command in CMS subset. When the files have been read onto the other user's disk, you can erase them from your disk. Then, return to edit mode and issue the FILE subcommand.

3. In CMS subset, erase the original disk file (if it existed), then return to edit mode and file the copy that you are editing. You should not use this method unless absolutely necessary, since any unexpected problems may result in the loss of both the disk file and the copy.

After you use the FILE subcommand to write the file onto disk, you should continue erasing any files you no longer need.

Summary of CMS EDIT Subcommands

The EDIT subcommands, and their formats, are shown in Figure A-4 . Refer to the *VM/SP CMS Command and Macro Reference* for complete details.

Subcommand Format	Function
ALTER	Scans the next n records of the file, altering the specified character, either once in each line or for all occurrences in the line.
AUTOSAVE	Automatically saves the file on disk after the indicated number of lines have been processed.
BACKWARD	Points the current line pointer to a line above the line currently pointed to.
BOTTOM	Makes the last line of the file the current line.
CASE	Indicates whether translation to uppercase is to be done, or displays the current status.
CHANGE	Changes string1 to string2 for n records or to EOF, either for the first occurrence in each line or for all occurrences.
CMS	Enters CMS subset command mode.
DELETE	Deletes n lines or to the end of the file (*).
DOWN	Points to the nth line from the current line.
DSTRING	Deletes all lines from the current line down to the line containing the indicated string.
FILE	Saves the file being edited on disk or changes its identifiers. Returns to CMS.
FIND	Searches for the given line.
FMODE	Resets or displays the filemode.
FNAME	Resets or displays the filename.
FORMAT	Switches the 3270 terminal between display mode and line mode. (3270 only)
FORWARD	Points to the nth line after the current line.
GETFILE	Inserts a portion or all of the specified file after the current line.
IMAGE	Expands text into line images or displays current line settings.
INPUT	Inserts a line in the file or enters input mode.
LINEMODE	Sets or displays current setting of line-number editing.
LOCATE	Scans file from next line for first occurrence of 'string'.
LONG	Enters long error message mode.
NEXT	Points the the nth line down from the current line.

Figure A-4 (Part 1 of 3). Summary of CMS EDIT Subcommands and Macros

Subcommand Format	Function
OVERLAY	Replaces all or part of the current line.
PRESERVE	Saves the current mode settings.
PROMPT	Sets or displays line number increment. Initial setting is 10.
QUIT	Terminates edit session with no updates incorporated since last save request.
RECFM	Sets or displays record format for subsequent files.
RENUM	Recomputes line numbers for VSBASIC and FREEFORT source files.
REPEAT	Executes the following OVERLAY subcommand n times.
REPLACE	Replaces the current line or deletes the current line and enters input mode.
RESTORE	Restores editor settings to values last preserved.
RETURN	Returns to edit environment from CMS subset.
REUSE	Stacks (LIFO) the last EDIT subcommand that does not start with REUSE or the question mark (?) and then executes any given EDIT subcommand.
SAVE	Saves the file on disk and stays in the edit environment.
SCROLL	Displays a number of screens of data above or below the current line (3270 only).
SERIAL	Turns serialization on or off in column 73 through 80.
SHORT	Enters short error message mode.
STACK	Stacks data lines or EDIT subcommands in the console input stack.
TABSET	Sets logical tab stops.
TOP	Moves the current line pointer to the null line at the top of the file.
TRUNC	Sets or displays the column of truncation. An asterisk (*) indicates the logical record length.
TYPE	Displays m lines beginning with the current line. Each line may be truncated to n characters.
UP	Moves the current line pointer toward the top of the file.
VERIFY	Sets, displays, or resets verification. An asterisk (*) indicates the logical record length.
{X|Y}	Assigns to X or Y the given EDIT subcommand or executes the previously assigned subcommand n times.
ZONE	Sets or displays the columns between which editing is to take place.
?	Displays the last EDIT subcommand, except = or ?.
nnnnn or nnnnnnnn	Locates the line specified by the given line number and inserts text, if given.
$DUP	Duplicates the current line n times. $DUP is an EDIT macro.
$MOVE	Moves up n lines or down m lines. $MOVE is an EDIT macro.

Figure A-4 (Part 2 of 3). Summary of CMS EDIT Subcommands and Macros

Appendix B. The CMS EXEC Processor

The CMS EXEC Processor

A CMS EXEC processor is a CMS file that contains executable statements. The statements may be CMS or CP commands or EXEC control statements. The execution can be conditionally controlled with additional EXEC statements, or it may contain no EXEC statements at all. In its simplest form, an EXEC file may contain only one record, have no variables, and expect no arguments to be passed to it. In its most complex form, it can contain thousands of records and may resemble a program written in a high-level programming language. As a CMS user, you should become familiar with the EXEC processor and use it often to tailor CMS commands to your own needs, as well as to create your own commands.

The following is an example of a simple EXEC procedure that might be named RDLINKS EXEC:

```
CP LINK DEWEY 191 291 RR DEWEY
CP LINK LIBRARY 192 292 RR DEWEY
ACCESS 291 B/A
ACC 292 C/A
```

When you enter:

```
rdlinks
```

each command line contained in the file RDLINKS EXEC is executed.

You could also create an EXEC procedure that functions like a cataloged procedure, and set it up to receive an argument, so that it executes somewhat differently each time you invoke it. For example, a file named ASM EXEC contains the following:

```
ASSEMBLE &1
PRINT &1 LISTING
LOAD &1
START
```

If you invoke the EXEC specifying the name of an assembler language source file, such as:

```
asm myprog
```

the procedure executes as follows:

```
ASSEMBLE MYPROG
PRINT MYPROG LISTING
LOAD MYPROG
START
```

The variable &1 in the EXEC file is substituted with the argument you enter when you execute the EXEC. As many as 30 arguments can be passed to an EXEC in this manner; the variables thus set range from &1 through &30.

Creating EXEC Files

EXEC files can be created with the CMS editors, by punching cards, or by using CMS commands or programs. When you create a file with the editor, records are, by default, variable-length with a logical record length of 80 characters. EXEC can process variable-length files of up to 130 characters. To create a variable-length EXEC file larger than 80 characters, use the LRECL option of the EDIT command:

```
edit new exec a (lrecl 130
```

To convert a variable-length file to a fixed-length file, you can edit the EXEC file and issue the subcommand:

```
recfm f
```

Or, you can use the COPYFILE command:

```
copyfile old exec a (recfm f
```

If you use fixed-length EXEC files, you should be aware that the EXEC interpreter only processes the first 72 characters of each record in a fixed-length file, regardless of the record length. You can, however, enter command or data lines that are longer than 72 characters to be processed by using the &BEGSTACK, &BEGTYPE, &BEGPUNCH, and &BEGEMSG control statements preceding the line(s) you want to be processed. If you specify &BEGPUNCH ALL, EXEC processes lines up to 80 characters long; if you specify &BEGTYPE ALL, &BEGSTACK ALL, or &BEGEMSG ALL, EXEC processes lines up to 130 characters.

In variable-length EXEC files, there are no such restrictions; lines up to 130 characters are processed in their entirety.

Two CMS commands create EXEC files. One is LISTFILE, which can be invoked with the EXEC option; it creates a file named CMS EXEC. The uses of CMS EXEC files are discussed under the heading "CMS EXECs and How To Use Them." The CMS/DOS command LISTIO creates an EXEC file named $LISTIO EXEC, which creates records for each of the system and programmer logical unit assignments. The LISTIO command and the $LISTIO EXEC are described in Chapter 10, "Developing VSE Programs Under CMS" on page 10-1.

Invoking EXEC Files

EXEC procedures are invoked when you enter the filename of the EXEC file. You can precede the filename on the command line with the CMS command, EXEC. For example:

```
exec test type list
```

where TEST is the filename of the EXEC file and TYPE and LIST are arguments (&1 and &2) you are passing to the EXEC. For example, an EXEC named PREPEDIT would be executed when you entered either:

```
prepedit newfile replace
-- or --
exec prepedit newfile replace
```

You must precede the EXEC filename with the EXEC command when:

- You invoke an EXEC from within another EXEC.
- You invoke an EXEC from a program.
- You have the implied EXEC function set off for your virtual machine.

The implied EXEC function is controlled by the SET command. If you issue the command:

```
set impex off
```

then you must use the EXEC command to invoke an EXEC procedure. The default setting is ON; you almost never need to change it.

An EXEC procedure having a synonym defined for it can be invoked by its synonym if the implied EXEC (IMPEX) function is on. However, within an EXEC procedure, only the EXEC filename can be used. A synonym is not recognized within an EXEC since the synonym tables are not searched during EXEC processing.

There is one EXEC file that you never have to specifically invoke. This is a PROFILE EXEC, which is automatically executed after you load CMS, when your A-disk is accessed. PROFILE EXECs are discussed next.

PROFILE EXECs

A PROFILE EXEC must have a filename of PROFILE. It can contain the CP and CMS commands you normally issue at the start of every terminal session. For example:

- Commands that describe your terminal characteristics, such as:

  ```
  CP SET LINEDIT ON
  SET BLIP *
  SET RDYMSG SMSG
  SYNONYM MYSYN
  ```

- Commands that spool your printer and punch for particular classes or characteristics:

  ```
  CP SPOOL E CLASS S HOLD
  ```

- Commands to initialize macro and text libraries that you commonly use:

  ```
  GLOBAL MACLIB OSMACRO CMSLIB
  GLOBAL TXTLIB PRIVLIB
  ```

- Commands to access disks that are a permanent part of your configuration:

  ```
  ACCESS 196 B
  ```

A PROFILE EXEC file that contains all of these commands might look like this:

```
&CONTROL OFF
CP SET LINEDIT ON
CP SPOOL E CLASS S HOLD
SET RDYMSG SMSG
SET BLIP *
SYNONYM MYSYN
GLOBAL MACLIB OSMACRO CMSLIB
GLOBAL TXTLIB PRIVLIB
ACCESS 196 B
```

&CONTROL OFF is an EXEC control statement that specifies that the CP and CMS command lines are not to be displayed on your terminal before they execute.

A PROFILE EXEC can be as simple or as complex as you require. As an EXEC file, it can contain any valid EXEC control statements or CMS commands. The only thing that makes it special is its filename, PROFILE, which causes it to be executed the first time you press the Return key after loading CMS.

Executing Your PROFILE EXEC

Usually, the first thing you do after loading CMS is to type a CMS command. When you press the Return key to enter this command or if you enter a null line, CMS searches your A-disk for a file with a filename of PROFILE and a filetype of EXEC. If such a file exists, it is executed before the first CMS command you enter is executed. Because you do not do anything special to cause your PROFILE EXEC to execute, you can say that it executes "automatically."

You can prevent your PROFILE EXEC from executing automatically by entering:

```
access (noprof)
```

as the first CMS command after you IPL CMS. You can enter:

```
profile
```

at any time during a CMS session to execute the PROFILE EXEC, if you had accessed your A-disk without it, or if you had made changes to it and wanted to execute it, or if you had changed your virtual machine.

CMS EXECs and How To Use Them

A file named CMS EXEC is created when you use the EXEC option of the LISTFILE command; for example:

```
listfile pr* document a (exec
```

The usual display that results from this LISTFILE command is a list of all the files on your A-disk with a filetype of DOCUMENT that have filenames beginning with the characters "PR." CMS, however, creates a CMS EXEC file that contains a record for each file that would be listed. The records are in the format:

```
&1 &2 filename filetype  filemode
```

Column 1 is blank. Now, if you have the following files on your A-disk:

```
PRFILE1 DOCUMENT
PRFILE2 DOCUMENT
PRFILE3 DOCUMENT
PRFILE4 DOCUMENT
```

The CMS EXEC file would contain the records:

```
&1 &2 PRFILE1  DOCUMENT A1
&1 &2 PRFILE2  DOCUMENT A1
&1 &2 PRFILE3  DOCUMENT A1
&1 &2 PRFILE4  DOCUMENT A1
```

In the preceding lines, &1 and &2 are variables that can receive values from arguments you pass to the EXEC when you execute it. For example, if you execute this CMS EXEC by issuing:

```
cms disk dump
```

the EXEC interpreter substitutes, on each line, the variable &1 with the DISK and the variable &2 with DUMP and executes the commands:

```
DISK DUMP PRFILE1 DOCUMENT A1
DISK DUMP PRFILE2 DOCUMENT A1
DISK DUMP PRFILE3 DOCUMENT A1
DISK DUMP PRFILE4 DOCUMENT A1
```

You can use this technique to transfer a number of files to another user. You should remember to spool your punch with the CONT option before you execute the EXEC, so that all of the files are transferred as a single spool file; for example:

```
cp spool d cont library
```

Then, after executing the EXEC file, close the punch:

```
cp spool d nocont close
```

If you pass only one argument to your CMS EXEC file, the variable &2 is set to a null string. For example:

```
cms erase
```

executes as:

```
ERASE PRFILE1   DOCUMENT A1
ERASE PRFILE2   DOCUMENT A1
ERASE PRFILE3   DOCUMENT A1
ERASE PRFILE4   DOCUMENT A1
```

You could also use a CMS EXEC to obtain a listing of files on a virtual disk. If you want, you can use one of the other LISTFILE command options with the EXEC option to get more information about the files listed. For example:

```
listfile * * a (exec date
```

produces a CMS EXEC that contains, in addition to the filename, filetype, and filemode of each file listed, the file format and size, and date information. You can then use the PRINT command to obtain a printed copy:

```
print cms exec
```

Before printing this file, you may want to use the SORT command to sort the list into alphabetic order by filename, by filetype, or both; for example:

```
sort cms exec a cmssort exec a
```

When you are prompted to enter sort fields, you can enter:

```
1 25
```

The file CMSSORT EXEC that is created contains a completely alphabetical list.

Modifying CMS EXECs

A CMS EXEC is like any other CMS file; you can edit it, erase it, rename it, or change it. If you have created it to catalog a particular group of files, you might want to rename it; each time you use the LISTFILE command with the EXEC option a CMS EXEC is created, and any old CMS EXEC is erased. To rename it, you can use the CMS RENAME command, or, if you are editing it, you can rename it when you file it:

```
edit cms exec
input &control off
file prfile exec
```

You might also want to edit a CMS EXEC to provide it with more numeric variables; for example:

```
edit cms exec
input &control off
input cp spool printer class s cont
change /a1/a1   &3 &4 &5 &6/ *
.
.
.
input cp spool printer nocont
input cp close printer
file prfile exec
prfile print % (cc
```

When this EXEC is executed, the variable &1 is substituted with PRINT, the variable &2 is set to a null string (the special character % indicates that you are not passing an argument to it), and &3 and &4 are set to the PRINT command option (CC, so that the files in the EXEC print with carriage control.

The substitution goes as follows:

&1	&2	&3	&4
PRINT	null	(	cc

The CP commands that are inserted ensure that the files print as a single spool file, and not individually.

Summary of the CMS EXEC Language Facilities

The CMS EXEC processor, or interpreter, recognizes keywords that begin with the special character ampersand (&). Keywords may indicate:

- Control statements
- Built-in functions
- Special variables
- Arguments

You may also define your own variables in an EXEC file; the CMS EXEC interpreter can process them as long as they begin with an ampersand. The following pages briefly discuss the kinds of things you can do with an EXEC, introduce you to the control statements, built-in functions, and special variables, and give some examples of how to use the CMS EXEC processor. For specific information on the format and usage rules for any EXEC statement or variable, consult the *VM/SP CMS Command and Macro Reference.*

In general the following rules apply to entering lines into an EXEC procedure:

1. Most input lines (with a few exceptions) are scanned during execution of the EXEC. Every word on a line is padded or truncated to fit into an eight-character "token." So, for example, if you enter the EXEC control statement:

   ```
   &type today is wednesday
   ```

   when this EXEC is executed, the line is displayed at your terminal:

   ```
   TODAY IS WEDNESDA
   ```

   The lines that are not tokenized are those that begin with an * (and are considered comments), and those that follow an &BEGEMSG, &BEGPUNCH, &BEGSTACK, or &BEGTYPE control statement, up to an &END statement.

2. You can enter input lines beginning in any column. The only time that you must enter an EXEC line beginning in column 1 is when you are using the &END control statement to terminate a series of lines being punched, stacked, or typed.

Arguments and Variables

Most EXEC processing is contingent on the value of variable expressions. A variable expression in an EXEC is a symbol that begins with an ampersand (&). When the EXEC interpreter processes a line and encounters a variable symbol, it substitutes the variable with a predefined value, if the symbol has been defined. Symbols can be defined in three ways:

1. when passed as arguments to the EXEC,

2. by assignment statements,

3. interactively, as a result of a &READ ARGS or &READ VARS control statement.

You can pass arguments to EXEC files when you invoke them. Each argument you enter is assigned a variable name: the first argument is &1, the second is &2, the third is &3, and so on. You can assign values for up to 30 variables this way. For example, if an EXEC is invoked:

```
scan alpha 2 notype print
```

the variable &1 has a value of ALPHA, the variable &2 has a value of 2, &3 is NOTYPE and &4 is PRINT. These values remain in effect until you change them.

You can test the arguments passed in several ways. The special variable &INDEX contains the number of arguments received. Using the example SCAN ALPHA 2 NOTYPE PRINT, the statement:

```
&IF &INDEX EQ 4 &GOTO -SET
```

would be true, since four arguments were entered, so a branch to the label -SET is taken.

You can change the values of arguments or assign values using the &ARGS control statement. For example:

```
&IF &INDEX EQ 0 &ARGS A B C
```

assigns the values A, B, and C to the variables &1, &2, and &3 when the EXEC is invoked without any arguments.

Use the &READ ARGS control statement to enter arguments interactively. For example, if your EXEC file contains the line:

```
&READ ARGS
```

when this line is executed, the EXEC issues a read to your virtual machine so that you can enter up to 30 arguments, to be assigned to the variables &1, &2, and so on.

The words that form an executable statement are searched for the names of EXEC variables. These variables are replaced by their values. This is done according to the following steps:

1. Each word is inspected for ampersands, starting with the rightmost character of the word and proceeding to the left.

2. If an ampersand is found, then it, with the rest of the word to the right, is taken as the name of an EXEC variable and replaced (in the word) by its value. This may increase or decrease the length of the word. Initially, all variables have a null value, except:

   a. The variables that represent the EXEC control words and predefined functions, that are initialized to their own names (for example, the value of "&IF" is "&IF").

   b. The EXEC arguments, and the other predefined variables.

3. Inspection resumes at the next character to the left, and the procedure is repeated from step 2 above, until the word is exhausted.

Assignment Statements

User-defined variable names begin with an ampersand (&) and contain up to seven additional characters. These variables can contain numeric or alphameric data. You define and initialize EXEC variables in assignment statements. In an assignment statement, the first data item starts with an ampersand (&) and the second data item is an equal sign (=). The value of the expression on the right side of the equal sign is assigned to the variable named on the left of the equal sign. For example:

```
&A = 35
```

is an assignment statement that assigns the numeric value 35 to the variable symbol &A. A subsequent assignment statement might be:

```
&B = &A + 10
```

After this assignment statement executes, the value of &B would be 35 plus 10, or 45.

You can use the &READ control statement to assign variable names interactively. For example, when the statement:

```
&READ VARS &NAME &AGE
```

is executed, the EXEC issues a read to your virtual machine, and you can enter a line of data. The first two words, or tokens, you enter are assigned to the variable symbols &NAME and &AGE, respectively.

> **Note:** The data item immediately following the target of an assignment statement must be an equal sign (=) and not an EXEC variable that has the value of an equal sign. Conversely, if an equal sign is to be the first data item following an EXEC control word, then it must be specified as an EXEC variable that has the value of an equal sign and not as an equal sign; otherwise, the statement is interpreted as an assignment statement and the control word is thereafter treated as a variable.

Null Variables

If you use a variable name that has not been defined, the variable symbol is set to a null string by the EXEC processor when the statement is executed. For example, if you have entered only two arguments on the EXEC command line, then the statement:

```
&IF &3 EQ CONT &ERROR &CONTINUE
```

is interpreted:

```
&IF  EQ CONT &ERROR &CONTINUE
```

&ERROR and &CONTINUE are recognized by EXEC as control statements. Since &3 is undefined, however, it is replaced by blanks and the resulting line produces an error during EXEC processing. You can prevent the error, and allow for null arguments or variables, by concatenating some other character with the variable. A period is used most frequently:

```
&IF .&3 EQ .CONT &ERROR &CONTINUE
```

If &3 is undefined when this line is scanned, the result is:

```
&IF . EQ .CONT &ERROR &CONTINUE
```

which is a valid control statement line.

Built-in Functions and Special Variables

The EXEC built-in functions are similar to those of higher-level languages. You can use the EXEC built-in functions to define variable symbols in an EXEC procedure.

Figure B-1 summarizes the built-in functions. It shows, given the variable &A, the values resulting in a variable &B when a built-in function is used to assign its value. Notice that all of the built-in functions are used on the right-hand side of assignment statements. Only the &LITERAL built-in function can be used in control statements; for example:

```
&TYPE &LITERAL &A
```

Function	Usage	Example	&B
		&A = 123	
&CONCAT	Concatenates tokens into a single token.	&B = &CONCAT &A 55	12355
&DATATYPE	Assigns the data type (NUM or CHAR) to the variable.	&B = &DATATYPE &A	NUM
&LENGTH	Assigns the length of a token to a variable.	&B = &LENGTH &A	3
&LITERAL	Prohibits substitution of a variable symbol.	&B = &LITERAL &A	&A
&SUBSTR	Extracts a character string from a token.	&B = &SUBSTR &A 2 2	23

Figure B-1. Summary of CMS EXEC Built-in Functions

Flow Control in an EXEC

An EXEC is processed line by line if a statement is encountered that passes control to another line in the procedure, execution continues there and each line is, again, executed sequentially. You can pass control with an &GOTO control statement:

```
&GOTO -BEGIN
```

where -BEGIN is a label. All labels in EXEC files must begin with a hyphen, and must be the first token on a line. For example:

```
-LOOP
```

A label may have control statements or commands following it; for example:

```
-HERE &CONTINUE
```

which indicates that the processing is to continue with the next line, or

```
-END &EXIT
```

The &EXIT control statement indicates that the EXEC processor should terminate execution of the EXEC and return control to CMS. You can also specify a return code on the &EXIT control statement:

```
&EXIT 6
```

results in a "(00006)" following the "R" in the CMS ready message. If you invoke a CMS command from the EXEC, you can specify that the return code from the CMS command be used:

```
&EXIT &RETCODE
```

Since the &RETCODE special variable is set after each CMS command that is executed, you can test it after any command to decide whether you want execution to end. For example, you could use the &IF control statement to test it:

```
&IF &RETCODE NE 0 &EXIT &RETCODE
```

"&EXIT &RETCODE" places the value of the CMS return code in the CMS ready message. You could place a line similar to the above following each of your CMS command lines, or you could use the &ERROR control statement, that will cause an exit as soon as an error is encountered:

```
&ERROR &EXIT &RETCODE
```

or you could use the &ERROR control statement to transfer control to some other part of your EXEC:

```
  &ERROR &GOTO -CHECK
.
.
.
       -CHECK
.
.
.
```

Another way to transfer control to another line is to use the &SKIP control statement:

```
&SKIP 10
```

transfers control to a line that is 10 lines below the &SKIP line. You can transfer control above the current line as well:

```
&IF &X NE &Y &SKIP -3
```

Transferring control with &SKIP is faster, when an EXEC is executing, than it is with &GOTO, but modifying your EXEC files becomes more difficult, particularly when you add or delete many lines.

You can use combinations of &IF, &GOTO, and &SKIP to set up loops in an EXEC. For example:

```
&X = 1
&IF &X = 4 &GOTO -ENDPRT
PRINT FILE&X TEST A
&X = &X + 1
&SKIP -3
-ENDPRT
```

Or, you can use the &LOOP control statement:

```
&X = 1
&LOOP 2 &X > 3
PRINT FILE&X TEST
&X = &X + 1
-ENDPRT
```

In both of these examples, a loop is established to print the files FILE1 TEST, FILE2 TEST, and FILE3 TEST. &X is initialized with a value of 1 and then incremented within the loop. The loop executes until the value of &X is greater than 3. As soon as this condition is met, control is passed to the label -ENDPRT.

Comparing Variable Symbols and Constants

In an EXEC, you can test whether a certain condition is true, and then perform some function based on the decision. Some examples have already appeared in this section, such as:

```
&LOOP 3 &X EQ &Y
```

In this example, the value of the variable &X is tested for an equal comparison with the value of the variable &Y. The loop is executed until the condition (&X equal to &Y) is true.

The logical comparisons you can make are:

Condition	Mnemonic	Symbol
equal	EQ	=
not equal	NE	¬=
greater than	GT	>
less than	LT	<
greater than or equal to	GE	>=
less than or equal to	LE	<=

Figure B-2. Logical Comparisons You can Make in EXEC

When you are testing a condition in an EXEC file, you can use either the mnemonic or the symbol to represent the condition:

```
&IF &A LT &B &GOTO -NEXT
```

is the same as:

```
&IF &A < &B &GOTO -NEXT
```

Doing I/O With an EXEC

You can communicate with your terminal using the &TYPE and &READ control statements. Use &TYPE to display a line at your terminal:

```
&TYPE ASMBLNG &1 ASSEMBLE
```

When this line is processed, if the variable &1 has a value of PROG1, the line is displayed as:

```
ASMBLNG PROG1 ASSEMBLE
```

Use the &READ control statement when you want to be able to enter data, variables, or control statements into your EXEC file while it is executing. If you use it with an &TYPE statement, for example:

```
&TYPE DO YOU WANT TO CONTINUE ?
&READ VARS &ANS
```

you could test the variable &ANS in your EXEC to find out how processing is to continue.

The &BEGTYPE control statement can be followed by a sequence of lines you want to be displayed at the terminal. For example, if you want to display ten lines of data, instead of using ten &TYPE control statements, you could use:

```
&BEGTYPE
line1
line2
.
.
.
line10
&END
```

The &END control statement indicates the end of the lines to be typed. You can also use the &BEGTYPE control statement when you want to type a line that contains a word with more than eight characters in it; for example:

```
&BEGTYPE
TODAY IS WEDNESDAY
&END
```

The EXEC interpreter, however, does not perform substitutions on lines entered this way. The lines:

```
&A = DOG
&BEGTYPE
MY &A IS NAMED FIDDLEFADDLE
&END
```

result in the display:

```
MY &A IS NAMED FIDDLEFADDLE
```

You must use the &TYPE statement when you want to display variable data; you must use the &BEGTYPE control statement to display words with more than eight characters.

To type null or blank lines at your terminal (to make output readable, for example), you can use the &SPACE control statement:

```
&SPACE 5
```

Using Your Virtual Card Punch

You can punch lines of tokens into your virtual card punch with the &PUNCH control statement:

```
&PUNCH &NAME &TOTAL
```

When you want to punch more than one line of data, or a line that contains a word of more than eight characters in it, you should use the &BEGPUNCH control statement preceding the lines you want to punch, and follow them with an &END statement. The EXEC processor does not interpret these lines, however, so any variable symbols you enter on these lines are not substituted.

When you punch lines from an EXEC procedure what you are actually doing is creating a file in your virtual card punch. To release the file for processing, you must close the punch:

```
cp close punch
```

The destination of the file depends on how you have spooled your punch. If you have spooled it to yourself, the file is placed in your virtual card reader, and you can read it onto a virtual disk using the READCARD command.

Stacking Lines

The EXEC control statements &STACK and &BEGSTACK allow you to stack lines in your program stack, to be executed as soon as a read occurs in your virtual machine. Stacking is useful when you use commands that require responses, for example, the SORT command:

```
&STACK 1 20
SORT INFILE FILE A OUTFILE FILE A
```

When the SORT command is executed, a prompting message is issued, the virtual machine read occurs, and the response that you have stacked is read. If you do not stack a response to this command, your EXEC does not continue processing until you enter the response from your terminal.

In the above example of the SORT command, you can suppress the prompting message by issuing either the SET CMSTYPE HT command or &STACK HT immediately before the SORT command. Restore normal terminal operations by placing either a SET CMSTYPE RT command or &STACK RT after the SORT command.

Stacking is useful in creating edit macros or in editing files from EXEC procedures.

> **Note:** &STACK HT and SET CMSTYPE HT create the same effect when interpreted by the CMS EXEC processor. Similarly, &STACK RT and SET CMSTYPE RT are equivalent for the EXEC 2 processor. However, when using EXEC 2, the commands &STACK HT and &STACK RT will cause the characters "HT" and "RT" to be placed in the program stack but will not affect the console output. Unless these characters are part of a program or cleared from the stack, you will receive an "UNKNOWN CP/CMS COMMAND" error message when they are read from the stack.

Monitoring EXEC Procedures

Two EXEC control statements, &CONTROL and &TIME, control how much information is displayed at your terminal while your EXEC file is executing. This display is called an execution summary.

Since you do not usually receive a CMS ready message after the execution of each CMS command in an EXEC, you do not receive the timing information that is provided with the ready message. If you want this timing information to appear, you can specify:

```
&TIME ON
```

or you can type the CPU times at particular places by using:

```
&TIME TYPE
```

The &CONTROL control statement allows you to specify whether certain lines or types of information are displayed during execution. By default, CP and CMS commands are displayed before they are executed. If you do not wish to see them displayed, you can specify:

```
&CONTROL OFF
```

You might find it useful, when you are debugging your EXECs, to use:

```
&CONTROL ALL
```

When you use this form, all EXEC statements, as well as all CP and CMS commands, are displayed and you can see the variable substitutions being performed and the branches being taken in a procedure.

Summary of CMS EXEC Control Statements

Figure B-3 summarizes CMS EXEC control statements.

Control Statement	Function
&variable	Assigns a value to the symbol specified by &variable; the equal sign must be preceded and followed by a blank.
&ARGS	Redefines the variable symbols &1, &2... with the values of &arg1, &arg2, ..., and resets the variable &INDEX.
&BEGEMSG	Displays the following lines as CMS error messages, without scanning them.
&BEGPUNCH	Punches the following lines in the virtual card punch, without scanning them.
&BEGSTACK	Stacks the following lines in the terminal input buffer, without scanning them.
&BEGTYPE	Displays the following lines at the console, without scanning them.
&CONTINUE	Provides a branch address for &ERROR, &GOTO, and other conditional branching statements.
&CONTROL	Sets, until further notice, the characteristics of the execution summary of the EXEC, which is displayed at the console.
&EMSG	Displays a line of tokens as a CMS error message.
&END	Terminates a series of lines following an &BEGEMSG, &BEGPUNCH, &BEGSTACK, or &BEGTYPE control statement.
&ERROR	Executes the specified whenever a CMS command returns a nonzero return code.
&EXIT	Exits from the EXEC file with the given return code.
&GOTO	Transfers control to the top of the EXEC file, to the given line, or to the line starting with the given label.
&HEX	Turns on or off hexadecimal conversion.
&IF	Executes the specified statement if the condition is satisfied.
&LOOP	Loops through the following n lines, or down to (and including) the line at label, for m times, or until the condition is satisfied.
&PUNCH	Punches the specified tokens to your virtual card punch.

Figure B-3 (Part 1 of 2). Summary of CMS EXEC Control Statements

Control Statement	Function
&READ	Reads lines from the terminal or from the console stack. ARGS assigns the tokens read to the variables &1, &2... VARS assigns the tokens read to the specified variable symbols.
&SKIP	Transfers control forward or backward a specified number of lines.
&SPACE	Displays blank lines at the terminal.
&STACK	Stacks a line in the terminal input stack.
&TIME	Displays timing information following the execution of CMS commands.
&TYPE	Displays a line at the terminal.

Figure B-3 (Part 2 of 2). Summary of CMS EXEC Control Statements

Summary of CMS EXEC Special Variables

Figure B-4 summarizes CMS EXEC special variables. In some cases you may assign your own value to the EXEC special variable. The column entries in the "Set by" column have the following meanings:

User Variables are assigned values by EXEC but you may modify them.

EXEC You may not modify these variables.

CMS You may assign a value to this variable but it is reset at the completion of each CMS command.

Variable	Usage	Set by
&n	Arguments passed to an EXEC are assigned to the variables &1 through &30.	User
&* and &$	Test whether all (&*) or any (&$) of the arguments passed to EXEC have a particular value.	EXEC
&DISKx	Indicates whether the disk access at mode 'x' is a CMS OS, or DOS disk, or not accessed (CMS, OS, DOS, or NA).	User
&DISK*	Contains the mode letter of the first read/write disk in the CMS search order, or NONE if no read/write disk is accessed.	User
&DISK?	Contains the mode letter of the read/write disk with the most available space or NONE, if no read/write disk is accessed.	User
&DOS	Indicates whether or not the CMS/DOS environment is active (ON or OFF).	User
&EXEC	Contains the filename of the EXEC file currently being executed.	EXEC
&GLOBAL	Has a value ranging from 1 to 19, to indicate the recursion (nesting) level of the EXEC that is currently executing.	EXEC
&GLOBALn	The variables &GLOBAL1 through &GLOBAL9 can contain integral numeric values, and can be passed among different recursion levels. If not explicitly set, the variable will have a value of 1.	User
&INDEX	Contains the number of arguments passed to the EXEC on the command line or the number of arguments entered as a result of an &ARGS or &READ ARGS control statement.	EXEC
&LINENUM	Contains the current line number in the EXEC.	EXEC
&READFLAG	Indicates whether (STACK) or not (CONSOLE) there are lines stacked in the terminal input buffer (console stack).	EXEC
&RETCODE	Contains the return code from the most recently executed CMS command.	CMS
&TYPEFLAG	Indicates whether (RT) or not (HT) output is being displayed at the console.	EXEC

Figure B-4 (Part 1 of 2). CMS EXEC Special Variables

Variable	Usage	Set by
&0	Contains the name of the EXEC file.	User

Figure B-4 (Part 2 of 2). CMS EXEC Special Variables

Building CMS EXEC Procedures

This section discusses various techniques that you can use when you write CMS EXEC procedures. The examples are intended only as suggestions. You should not feel that they represent either the only way or the best way to achieve a particular result. Many combinations and variations of control statements are possible; in most cases, there are many ways to do the same thing.

This section is called "Building CMS EXEC Procedures" because you will often find that once you have created an EXEC procedure and begun to use it, you continually think of new applications or new uses for it. Using the CMS editor, you may quickly build the additions and make the necessary changes. You are encouraged to develop EXEC procedures to help you in all the phases of your CMS work.

> **Note:** If you are using EXEC 2, refer to *VM/SP EXEC 2 Reference* for detailed information.

What is a Token?

An executable statement is any line in an EXEC file that is processed by the EXEC interpreter, including:

- CMS command lines
- EXEC control statements
- Assignment statements
- Null lines

Executable statements may appear by themselves on a line or as the object of another executable statement, for example in an &IF or &LOOP control statement. If you want to execute CP commands or other EXEC procedures in an EXEC, you must use the CP and EXEC commands, respectively. CP commands are passed directly to CP for processing.

All executable statements in an EXEC are scanned by the CMS scan routine. This routine converts each word (words are delimited by blanks and parentheses) into an eight-character quantity called a token. If a word contains more than eight characters, it is truncated on the right. If it contains fewer than eight characters, it is padded with blanks. When a parenthesis appears on the line, it is treated both as a delimiter and as a token. For example, the line:

```
:TYPE WHAT IS YOUR PREFERENCE (RED|BLUE)?
```

scans as follows:

```
:TYPE WHAT IS YOUR PREFEREN ( RED|BLUE ) ?
```

After a line has been scanned, each token is scanned for ampersands and substitutions are performed on any variable symbols in the tokens before the statement is executed. After elimination of any null variables, the statement may contain a maximum of 32 tokens.

Nonexecutable statements are lines that are not processed by the EXEC interpreter, that is, comment lines (those that begin with an *), and data lines following an &BEGEMSG, &BEGPUNCH, &BEGSTACK, or &BEGTYPE control statement. Since these lines are not scanned, words are not truncated, and tokens are neither formed nor substituted.

Since all executable statements in an EXEC are scanned, and the data items are treated as tokens, the term "token" is used throughout this section to describe data items before and after scanning. The *VM/SP CMS Command and Macro Reference*, which contains the formats and descriptions of the EXEC control statements, uses this convention as well. Therefore, as you create your EXEC procedures, you may think of the items that you enter on an EXEC statement as tokens, since that is how they are used by the EXEC interpreter.

Variables

To make the best use of the CMS EXEC facilities, you should have an understanding of how the EXEC interpreter performs substitutions on variable symbols contained in tokens. Some examples follow. For each example, the input lines are shown as they would appear in an EXEC file and as they would appear after being interpreted and executed by EXEC. Notes concerning substitution follow each example.

Simple Substitution: Most of the EXEC examples in this publication contain variable symbols that result in one-for-one substitution. Most of your variables, too, will have a similar relationship.

Lines	After Substitution
`&X = 123`	`&X = 123`
`&TYPE &X`	`&TYPE 123`

The EXEC interpreter accepts the variable symbol &X and assigns it the value 123. In the second statement, &X is substituted with this value, and the control statement &TYPE is recognized and executed.

Lines	After Substitution
`&Y = 456`	`&Y = 456`
`&Z = &Y`	`&Z = 456`

The symbol &Y is assigned a value of 456. In the second statement, the symbol &Y is substituted with this value, and this value is assigned to &Z.

Subscripts for Variables: Since each token is scanned more than once for ampersands, you can simulate subscripts by using two variable values in the same token.

Lines	After Substitution
`&1 = ALPHA`	`&1 = ALPHA`
`&2 = BETA`	`&2 = BETA`
`&INDEX1 = 1`	`&INDEX1 = 1`
`&TYPE &&INDEX1`	`&TYPE ALPHA`
`&INDEX1 = 2`	`&INDEX1 = 2`
`&TYPE &&INDEX1`	`&TYPE BETA`

In the statement &TYPE &&INDEX1, the token &INDEX1 is scanned the first time, and the value &INDEX1 is substituted with the value 1. The token now contains &1, which is substituted with the value ALPHA on a second scan. When the value of &INDEX1 is changed to 2, the value of &&INDEX1 also changes.

Lines	After Substitution
`&I = 2`	`&I = 2`
`&X&I = 5`	`&X2 = 5`
`&I = 1`	`&I = 1`
`&X&I = 2`	`&X1 = 2`
`&X = &X&I + &X&X&I`	`&X = 2 + 5`

In the statement &X&I = 5, analysis of the first token results in the substitution of the symbol &I with the value of 2. The symbol &X2 is assigned a value of 5.

The value of &I is changed to 1, and the symbol &X1 is assigned a value of 2.

In the last statement, &X = &X&I + &X&X&I, the value of &I in the token &X&I is replaced with 1, then the symbol &X1 is substituted with its value, which is 2. The token &X&X&I is substituted after each of three scans: &I is replaced with the value 1, to yield the token &X&X1. The symbol &X1 has the value of 2, so the token is reduced to &X2, which has a value of 5.

Compound Variable Symbols: Variable symbols may also be combined with character strings.

Lines	After Substitution
`&X = BEE`	`&X = BEE`
`&TYPE HONEY&X`	`&TYPE HONEYBEE`
`&TYPE ABUMBLE&X`	`&TYPE ABUMBLE`

In the above example, the first symbol encountered in the scan of HONEY&X is &X, which is substituted with the value &BEE. In the second &TYPE statement, the X is truncated when the line is scanned; the symbol & is an undefined symbol and is therefore set to blanks.

Lines	After Substitution
`&X = HONEY`	`&X = HONEY`
`&Y = BEE`	`&Y = BEE`
`&TYPE &X&Y`	`&TYPE`

In the above example, after the symbol &Y is substituted with the value BEE, the token contains the symbol &XBEE, which is an undefined symbol, so the symbol is discarded.

Lines	After Substitution
`&123 = ABCDE`	`&123 = ABCDE`
`&X = 12345678`	`&X = 12345678`
`&TYPE ABLE&&X`	`&TYPE ABLEABCD`

In this example, the substitution of &X in the token ABLE&&X results in the character string ABLE&12345678, which is truncated to eight characters, or ABLE&123. The scan continues, and &123 is substituted with the appropriate value, to result in ABCDE. The token is again truncated to eight characters.

Concatenation of tokens: The &CONCAT built-in function is used to concatenate two or more tokens.

Lines	After Substitution
&X = BB	&X = BB
&Y = &CONCAT AA &X CC	&Y = &CONCAT AA BB CC
&TYPE &Y	&TYPE AABBCC

In the above example, the substitution of &Y results in the character string &CONCATAABBCC. The scan continues with the concatenation, the result, AABBCC.

Substituting Literal Values: You might want an ampersand to appear in a token. You can use the &LITERAL built-in function to suppress the substitution of variable symbols in a token.

Lines	After Substitution
&9 = HELLO	&9 = HELLO
&A = &LITERAL &9	&A = &LITERAL &9
&TYPE &A	&TYPE &9

Because the value of &A was defined as a literal &9, no substitution is performed.

Lines	After Substitution
&TYPE = QUERY	&TYPE = QUERY
&TYPE BLIP	QUERY BLIP

In the above example, even though &TYPE is an EXEC keyword, it is assigned the value of QUERY, and substitution is performed when it appears on an input line. In this example, when it is substituted with its value, the result is a command line which is passed to CMS for processing.

Lines	After Substitution
&CONTROL = FIRST	&CONTROL = FIRST
&LITERAL &CONTROL ALL	&CONTROL ALL

In this example, &CONTROL is assigned a value as a variable symbol, but when it is preceded by the built-in function &LITERAL, the substitution is not performed, so EXEC processes it as a control statement.

Hexadecimal and Decimal Conversions: You can perform hexadecimal to decimal and decimal to hexadecimal conversions in an EXEC if you use the control statement &HEX ON.

Tokens of the form X'xxx, can be converted from hexadecimal to decimal and from decimal to hexidecimal. The conversion takes place according to the rules given below. These rules are in effect *only* if '&HEX ON' is in effect.

1. Hexadecimal-to-decimal conversion is performed in the assignment statement, and that is the *only.* place where it occurs.

   Example:

   ```
   &X = 100 + X'100
   ```

   This results in &X being set to 356 (100 + 256)

2. Decimal-to-hexadecimal conversion is performed whenever substitution is performed, *except* on the right-hand-side of an assignment.

   Example:

```
&STACK LIFO 100 X'100 X'15
&READ VARS &A &B &C
```

   This sets &A to 100, &B to 64, and &C to F.

3. No conversion is performed on the left-hand-side of an assignment statement. Instead, the quote is treated as an illegal character in a variable name.

4. Conversion errors occur if the conversion cannot be performed, either because the result is too large, or because the number contains invalid digits.

   Example:

```
&X = FFFFFFF
&Y = X'&X
```

   The result of the conversion of X'FFFFFFF' to decimal is larger than the maximum of 99999999 decimal.

   Note: No intermediate truncation occurs during conversion, as in the preceding example, where X'FFFFFFF contains 9 characters.

   Example:

```
&TYPE X'FFFF
```

   The conversion argument is expected to be a decimal number.

The following illustrates conversions with '&HEX ON' in effect:

EXEC Control Statements	**Result After Substitution**
`  &CONTROL ALL`	
`-E1 &HEX ON`	
`  &NUM = X'FFFFFF`	
`  &TYPE HEX X'&NUM = DEC &NUM`	`&NUM = 16777215`
	`&TYPE HEX FFFFFF`
	`  = DEC 16777215`
`-E2 &IF X'16777215 = X'&NUM &GOTO -E3`	`&IF 28F5C = FFFFFF`
	`  &GOTO -E3`
`  &TYPE &LITERAL X'16777215`	`&TYPE X'167772 NE X'&NUM`
`    NE &LITERAL X'&NUM`	
`  &TYPE X'16777215 NE X'&NUM`	`&TYPE 28F5C NE FFFFFF`
`-E3 &NUM = X'10`	`&NUM = 16`
`  &Y = &NUM + X'B`	`&Y = 16 + 11`
`  &TYPE &Y X'&Y`	`&TYPE 27 1B`
`-E4 &Y = X'&NUM`	`&Y = 22`
`  &Z = &CONCAT &LITERAL X'1 X'&NUM`	`&Z = &CONCAT X'1 22`
`  &HEX OFF`	`&HEX OFF`
`  &TYPE &Y &Z`	`&TYPE 22 X'122`
`  &HEX ON`	`&HEX ON`
`  &TYPE  &Y &Z`	`&TYPE 22 7A`

To suppress hexadecimal conversion during an EXEC procedure after having used it, you can use the CMS EXEC control statement:

```
&HEX OFF
```

so you can use tokens containing the characters X' without the EXEC processor converting them to hexadecimal.

Arguments

An argument in an CMS EXEC procedure is one of the special variable symbols &1 through &30 that are assigned values when the EXEC is invoked. For example, if the EXEC named LINKS is invoked with the line:

```
links viola ariel oberon
```

the tokens VIOLA, ARIEL, and OBERON are arguments and are assigned to the variable symbols &1, &2, and &3, respectively.

You can pass as many as 30 arguments to an EXEC procedure; thus the variable symbols you can set range from &1 to &30. These variables are collectively referred to as the special variable &n. Once these symbols are defined, they can be used and manipulated in the same manner as any other variable in an EXEC. They can be tested, displayed, changed, and, if they contain numeric quantities, used arithmetically.

```
&IF &2 EQ PRINT &GOTO -PR
&TYPE &1 IS AN INVALID ARGUMENT
&1 = 2
&CT = &1 + 100
```

The above examples illustrate some explicit methods of manipulating the &n variables. They can also be implicitly defined or redefined by two EXEC control statements: &ARGS and &READ ARGS.

An &ARGS control statement redefines all of the special &n variables. The statement:

```
&ARGS A B C
```

assigns the value of A, B, and C to the variables &1, &2, and &3 and sets the remaining variables, &4 through &30, to blanks.

You can also redefine arguments interactively by using the &READ ARGS control statement. When EXEC processes this statement, a read request is presented to your terminal, and the tokens you enter are assigned to the &n variables. For example, the lines:

```
&TYPE ENTER FILE NAME AND TYPE:
&READ ARGS
STATE &1 &2 *
```

request you to enter two tokens, and then treat these tokens as the arguments &1 and &2. The remaining variables &3 through &30 are set to blanks.

If you want to redefine specific &n variables, and leave the values of others intact, you can either redefine the individual variables in separate assignment statements, or use the variable symbol in the &ARGS or &READ ARGS statement. For example, the statement:

```
&ARGS CONT &2 &3 RETURN &5 &6 &7 &8 &9 &10
```

assigns new values to the variables &1 and &4, but does not change the existing values for the remaining symbols through &10.

If you need to set an argument or &n special variable to blanks, either on an EXEC command line or in an &ARGS or &READ ARGS control statement, you can use a percent sign (%) in its place. For example, the lines:

```
&ARGS SET QUERY % TYPE
&TYPE &1 &2 &3 &4
```

result in the display:

```
SET QUERY TYPE
```

The symbol &3 has a value of blanks, and as a null token, is discarded from the line.

Using the &INDEX Special Variable

The EXEC special variable, &INDEX, initially contains a numeric value corresponding to the number of arguments defined when the EXEC was invoked. The value of &INDEX is reset whenever an &ARGS or &READ ARGS control statement is executed.

&INDEX can be useful in many circumstances. If you create an EXEC that may expect any number of arguments, and you are going to perform the same operation for each, you might set a counter and use the value of &INDEX to test it. For example, an EXEC named PRINTX accepts arguments that are the filenames of ASSEMBLE files:

```
&CT = 1
&LOOP 2 &CT > &INDEX
PRINT &&CT ASSEMBLE
&CT = &CT + 1
```

In the preceding example, the token &&CT is substituted with &1, &2, and so on until all of the arguments entered on the PRINTX line are used.

You can also use &INDEX to test the number of arguments entered. If you design an EXEC to expect at least two arguments, the procedure might contain the statements:

```
&IF &INDEX LT 2 &GOTO -ERR1
.
.
.
-ERR1 &TYPE INVALID COMMAND LINE
&EXIT 1
```

In this example, if the EXEC is invoked with one or no arguments, an error message is displayed and the EXEC terminates processing with a return code of 1.

As another example, suppose you wanted to supply an EXEC with default arguments, which might or might not be overridden. If the EXEC is invoked with no arguments, the default values are in effect; if it is invoked with arguments, the arguments replace the default values:

```
&DISP = PRINT
&COUNT = 2
&IF &INDEX GT 2 &EXIT 1
&IF &INDEX EQ 0 &GOTO -GO
&COUNT = &1
&IF &INDEX = 2 &DISP = &2
-GO
```

Default values are supplied for the variables &DISP and &COUNT. Then, &INDEX is tested, and the variables are reset if any arguments were entered.

Checking Arguments

There are a number of tests that you can perform on arguments passed to a CMS EXEC. In some cases, you may want to test for the length of a specific argument or to test whether an argument is character data or numeric data. To perform these tests, you can use the EXEC built-in functions &LENGTH and &DATATYPE.

To use either &LENGTH or &DATATYPE, you must first assign a variable to receive the result of the function, and then test the variable. For example, to test whether an entered argument is five characters long, you could use the statements:

```
&LIMIT = &LENGTH &1
&IF &LIMIT GT 5 &EXIT &LIMIT
```

When these statements are executed, if the first argument (&1) is greater than five characters, the exit is taken, and the return code indicates the length of &1.

If you wish to check whether a number was entered for an argument, use the &DATATYPE function:

```
&STRING = &DATATYPE &2
&IF &STRING ¬= NUM &GOTO -ERR4
```

In this example, the second argument expected by the EXEC must be a numeric quantity. If it is not, a branch is taken to an error exit routine.

Often, you may create an EXEC that tests for specific arguments and then takes various paths, depending on the argument. For example:

```
&IF &2 = PRINT &GOTO -PR
&IF &2 = TYPE &GOTO -TY
&IF &2 = ERASE &GOTO -ER
&EXIT
```

In this EXEC, if the value of &2 is not PRINT, TYPE, or ERASE, or was not entered, the EXEC terminates processing.

&* and &$

There are two special EXEC keywords that you may use to test arguments passed in an EXEC. They are &* and &$, which can be used only in an &IF or an &LOOP control statement. They test the entire range of numeric variables &1 through &30, as follows:

***&$***: The special token &$ is interpreted as "any of the variables &1, &2, ..., &30."That is, if the value of any one of the numeric variables satisfies the established condition, then the &IF statement is considered to be true. The statement is false only when none of the variables fulfills the specified requirements.

As an example, suppose you want to make sure that some particular value is passed to the EXEC. You can check to see if any of the arguments satisfy this condition, as follows:

```
&IF &$ EQ PRINT &SKIP 2
&TYPE PARM LIST MUST INCLUDE PRINT
&EXIT
```

In this example, the path to the &TYPE statement is taken only when none of the arguments passed to the EXEC procedure equal the character string PRINT.

&*: The special token &* is interpreted as "all of the variables &1, &2, ..., &30."That is, if the value of each of the numeric variables satisfies the established condition, then the &IF statement is considered to be true. The statement is false when at least one of the variables fails to meet the specified requirements.

Use &* to test for the absence of an argument as follows:

```
&IF &* NE ASSEMBLE &EXIT 3
```

In this example, if an EXEC is invoked, and none of the arguments equals ASSEMBLE, the EXEC terminates with a return code of 3.

The tokens &* and &$ are set by arguments entered when an EXEC is invoked and reset when you issue an &ARGS or &READ ARGS control statement. If either &* or &$ is null because no arguments are entered, the &IF statement is considered a null statement, and no error occurs.

Execution Paths in a CMS EXEC

You have already seen examples of the &IF, &GOTO, &SKIP, and &LOOP control statements. A more detailed discussion on each of these statements and additional techniques for controlling execution paths in an EXEC procedure follow.

Labels in a CMS EXEC Procedure

In many instances, an execution control statement in an EXEC procedure causes a branch to a particular statement that is identified by a label. The rules and conventions for creating syntactically correct EXEC labels are:

- A label must begin with a hyphen (dash) and must have at least one additional character following the hyphen.
- Up to seven additional alphameric characters may follow the hyphen (with no intervening blanks). However, the sequence:

  ```
  &GOTO -PROBABLY
  .
  .
  .
  -PROBABLY
  ```

  executes successfully, because when each statement is scanned, the token -PROBABLY is truncated to the same eight-character token, -PROBABL.
- A label name may be the object of an &GOTO or &LOOP control statement.
- A label that is branched to must be the first token on a line. It may stand by itself, with no other tokens on the line, or it may precede an executable CMS command or CMS EXEC control statement. The following are examples of the correct use of labels:

```
&GOTO -LAB1
-LAB1
-LAB2 &CONTINUE
-CHECK &IF &INDEX EQ 0 &GOTO -EXIT
&IF &INDEX LT 5 &SKIP
-EXIT &EXIT 4
&TYPE &LITERAL &INDEX VALUE IS &INDEX
```

Conditional Execution with the &IF Statement

The main tool available to you for controlling conditional execution in a CMS EXEC procedure is the &IF control statement. The &IF control statement provides the decision-making ability that you need to set up conditional branches in your EXEC procedure.

One approach to decision-making with the &IF control statement is to compare two tokens, and perform some action based on the result of the comparison. When the comparison is specified true, the executable statement is executed. When the comparison is false, control passes to the next sequential statement in the EXEC procedure. An example of a simple &IF statement is:

```
&IF &1 EQ &2 &TYPE MATCH FOUND
```

If the comparand values are not equal, the statement &TYPE MATCH FOUND is not executed, and control passes to the next statement in the EXEC procedure. If the comparand values are equal, the statement &TYPE MATCH FOUND is executed before control passes to the next statement. &IF statements can also be used to establish a comparison between a variable and a constant. For example, if a terminal user could properly enter a YES or NO response to a prompting message, you could set up &IF statements to check the response as follows:

```
&READ ARGS
&IF &1 EQ YES &GOTO -YESANS
&IF &1 EQ NO &GOTO -NOANS
&TYPE &1 IS NOT A VALID RESPØNSE (MUST BE YES OR NO)
&EXIT
-YESANS
.
.
.
-NOANS
.
.
.
```

In this example, the branch to -YESANS is taken if the entered argument is YES; otherwise, control passes to the next &IF statement. The branch to -NOANS is taken if the argument is NO; otherwise, control passes to the &TYPE statement, which displays the entered argument in an error message and exits.

The test performed in an &IF statement need not be a simple test of equality between two tokens; other types of comparisons can be tested. The tests that can be performed and the symbols you can use to represent them are:

Symbol	Mnemonic	Meaning
=	EQ	A equals B
¬=	NE	A does not equal B

Symbol	Mnemonic	Meaning
<	LT	A is less than B
<=	LE	A is less than or equal to B (not greater than)
>	GT	A is greater than B
>=	GE	A is greater than or equal to B (not less than)

Compound &IF Statements

You can place multiple &IF control statements on one line, to test a variable for more than one condition. For example, the statement:

```
&IF &NUM GT 5 &IF &NUM LT 10 &TYPE O.K.
```

checks the variable symbol &NUM for a value greater than 5 and less than 10. If both of these conditions are satisfied, the &IF statement is true, and the &TYPE statement is executed. If either condition is false, then the &TYPE statement is not executed.

The number of &IF statements that may be nested is limited only by the following criteria:

CMS EXEC files	a maximum of 32 tokens for variable-length files. 72 characters for fixed-length records.
EXEC 2 files	the record length of the file.

Branching with the &GOTO Statement

The &GOTO control statement allows you to transfer control within your EXEC procedure:

- To a specified EXEC label somewhere in the EXEC file:

  ```
  &GOTO -TEST
  ```

  where -TEST is the label to which control is passed.

- To a particular line within the EXEC file. For example:

  ```
  &GOTO 15
  ```

  results in control being passed to statement 15 in the EXEC file.

- Directly to the top of the EXEC file. For example:

  ```
  &GOTO TOP
  ```

  passes control to the beginning of the EXEC procedure.

The &GOTO control statement can be coded wherever an executable statement is permitted in an EXEC procedure. One of its common uses is in conjunction with the &IF control statement. For example, in the statement:

```
&IF &INDEX EQ 0 &GOTO -ERROR
```

the branch to the statement labeled -ERROR is taken when the value of the &INDEX special variable is zero. Otherwise, control passes to the next sequential statement in the EXEC procedure.

An &GOTO statement can also stand alone as an EXEC control statement. When coded as such, it forces an unconditional branch to the specified location. For example, you might create an EXEC that has several execution paths, each of which terminates with an &GOTO statement leading to a common exit routine:

```
-PATH1 &CONTINUE
.
.
.
&GOTO -EXIT
-PATH2 &CONTINUE
.
.
.
&GOTO -EXIT
&PATH3 &CONTINUE
.
.
.
-EXIT &CONTINUE
```

You can use the &GOTO control statement to establish a loop. For example:

```
&GLOBAL1 = &GLOBAL1 + 1
&TYPE ENTER NUMBER:
&READ VARS &NEXT
&IF .&NEXT = . &GOTO -FINIS
&IF &GLOBAL1 = 2 &TOTAL = 0
&TOTAL = &TOTAL + &NEXT
&GOTO TOP
-FINIS
&TYPE TOTAL IS &TOTAL
```

In this EXEC example, the value of &GLOBAL1 is one initially and &TOTAL is set to zero the first time through the loop. All of the statements, through the &GOTO TOP statement, are executed repeatedly until a null line is entered in response to the prompting message. Then, the branch is taken to the label -FINIS and the total is typed.

Using the &GOTO Control Statement: When an EXEC procedure processes an &GOTO statement, and searches for a given label or line number, the scan begins on the line following the &GOTO statement, proceeds to the bottom of the file, then wraps around to the top of the file and continues to the line immediately preceding the &GOTO statement. If there are duplicate labels in an EXEC file, the first label encountered during the search is the one that is branched to.

If the label or line number is not found during the scan, EXEC terminates processing and displays the message:

```
ERROR IN EXEC FILE filename, LINE n - &SKIP or &GOTO ERROR
```

If the label or line number is found, control is passed to that location and execution continues.

Branching with the &SKIP Statement

The &SKIP control statement provides you with a second method of passing control to various points in an EXEC procedure. Instead of branching to a named or numbered location in an EXEC procedure, &SKIP passes control a specified number of lines forward or backward in the file.

You pass control forward in an EXEC by specifying how many lines to skip. For example, to handle a conditional exit from an EXEC procedure, you could code the following:

```
&IF &RETCODE EQ 0 &SKIP
&EXIT &RETCODE
```

where the &EXIT statement is skipped whenever the value of &RETCODE equals zero. If the value of &RETCODE does not equal zero, control passes out of the current EXEC procedure with a return code that is the nonzero value in &RETCODE. Note that when no &SKIP operand is specified, a value of 1 is assumed.

A succession of &SKIP statements can be used to perform multiple tests on a variable. For example, suppose an argument should contain a value from 5 to 10 inclusive:

```
&IF &1 LT 5 &SKIP
&IF &1 LE 10 &SKIP
&TYPE &1 IS NOT WITHIN RANGE 5-10
```

If the value of &1 is less than 5, control passes to the &TYPE control statement, which displays the erroneous value and an explanatory message. If the value of &1 is greater than or equal to 5, the next statement checks to see if it is less than or equal to 10. If this is true, then the value is between 5 and 10 inclusive, and execution continues following the &TYPE statement.

When you want to pass control to a statement that precedes the current line, determine how many lines backward you want to go, and code &SKIP with the desired negative value:

```
&VAL = 1
&TYPE &VAL
&VAL = &VAL + 1
&IF &VAL NE 10 &SKIP -2
```

In this EXEC, the &TYPE statement is executed repeatedly until the value of &VAL is 10, and then execution continues with the statement following the &IF statement.

Using Counters for Loop Control

A primary consideration in designing a portion of an EXEC procedure that is to be executed many times is how to control the number of executions. One way to control the execution of a sequence of instructions is to create a loop that tests and changes the value of a counter.

Before entering the loop, the counter is initialized to a value. Each time through the loop, the counter is adjusted (increased or decreased) toward a limit value. When the limit value is reached (the counter value is the same as the limit value), control passes out of the loop and it is not executed again. For example, the following EXEC initializes a counter based on the argument &1:

```
&IF &INDEX EQ 0 &EXIT 12
&TYPE COUNT IS &1
&1 = &1 - 1
&IF &1 GT 0 &SKIP -2
```

When this EXEC procedure is invoked, it checks that at least one argument was passed to it. If an argument is passed, it is assumed to be a number that indicates how many times the loop is to execute. Each time it passes through the loop, the value of &1 is decreased by 1. When the value of &1 reaches zero, control passes from the loop to the next sequential statement.

There are several ways of setting, adjusting, and testing counters. Some ways to set counters are by:

- Reading arguments from a terminal, such as:

  ```
  &READ VARS &COUNT1 &COUNT2
  ```

- Assigning an arbitrary value, such as:

  ```
  &COUNTER = 43
  ```

- Assigning a variable value or expression, such as:

  ```
  &COUNTS = &INDEX - 1
  ```

  Counter values can be increased or decreased by any increment or decrement that meets your purposes. For example:

  ```
  &COUNTEM = &COUNTEM - &RETCODE
  &COUNT1 = &COUNT + 100
  ```

Loop Control with the &LOOP Statement

A convenient way to control execution of a sequence of EXEC statements is with the &LOOP control statement. An &LOOP statement can be set up in four ways:

- To execute a particular number of statements a specified number of times. For example, the statement:

  ```
  &LOOP 3 2
  ```

  indicates that the three statements following the &LOOP statement are to be executed twice.

- To execute a particular number of statements until a specified condition is satisfied. For example:

  ```
  &LOOP 4 &X = 0
  ```

  The four statements following this statement are executed until the value of &X is 0.

- To execute the statements down to (and including) the statement identified by a label for a specified number of times. For example:

  ```
  &LOOP -ENDLOOP 6
  ```

  The statements between this &LOOP statement and the label -ENDLOOP are executed six times.

- To execute the statements down to (and including) the statement identified by a label until a specified condition is satisfied. In the following example:

  ```
  &LOOP -ENDLOOP &X = 0
  ```

  the statements are executed repeatedly until the value of &X is 0.

The numbers specified for the number of lines to execute and the number of times through the loop must be positive integers. You can use a variable symbol to represent the integer. If a label is used to define the limit of the loop, it must follow the &LOOP statement (it cannot precede the &LOOP statement).

In a conditional &LOOP statement, any variable symbols in the conditional phrase are substituted each time the loop is executed. For example, the statements:

```
&X = 0
&LOOP -END &X EQ 2
&X = &X + 1
-END &TYPE &X
```

are interpreted and executed as follows:

1. The variable &X is assigned a value of 0.

2. The &LOOP statement is interpreted as a conditional form; that is, to loop to -END until the value of &X equals 2. Since the value of &X is not 2, the loop is entered.

3. The variable &X is increased by 1 and is then displayed.

4. Control returns to the beginning of the loop, where &X is tested to see if it equals 2. Since it does not, the loop is executed again and 2 is displayed. The next time through the loop, when &X equals 2, control is passed to the EXEC statement immediately following the label -END.

When this EXEC procedure is executed, the following lines are displayed:

```
1
2
```

at which time the value of &X equals 2; the loop is not executed again.

Another example of a conditional loop is:

```
&Y = &LITERAL A&B
&LOOP 2 .&X EQ &LITERAL .&
&X = &SUBSTR &Y 2 1
&TYPE &X
```

These statements are interpreted and executed as follows:

1. The variable &Y is set to the literal value A&B.

2. The two statements following the &LOOP statement are to be executed until the value of &X is &.

3. The &SUBSTR built-in function is used to set the variable &X to the value of the second character in the variable &Y, which is a literal ampersand (&).

4. The ampersand is typed once, and the loop does not execute again because the condition that the value of &X be a literal ampersand is met.

Nesting CMS EXEC Procedures

If you want to use a CMS EXEC procedure within another CMS EXEC, you must use the EXEC command to execute it. For example, if you have the statement:

```
EXEC TEST
```

in an EXEC procedure, it invokes the EXEC procedure TEST. The procedure TEST EXEC executes independently of the other EXEC; the variables &1, &2 and so on are assigned values and the default settings for control statements such as &CONTROL and &HEX are reset. When TEST EXEC completes execution, control returns to the next line in the calling EXEC, where the values for variable symbols and EXEC settings are the same as when the TEST EXEC was invoked.

Passing Arguments to Nested Procedures: Variables in an EXEC file have meaning only within the particular procedure in which they are defined. There are two methods you can use to pass variable information to nested EXECs. One way is to pass arguments on the EXEC command line. For example, if the CHECK EXEC contains the line:

```
EXEC COUNTEM &ITEMCT &NUM
```

then the current values of &ITEMCT and &NUM are assigned to the variable symbols &1 and &2 in COUNTEM EXEC. (The values of &1 and &2 in CHECK EXEC do not change.)

You can also use the ten special variables &GLOBAL0 through &GLOBAL9. These variables can only contain integral numeric values; you cannot assign them character-string values. These variables can be used to set up arguments to pass to nested procedures, or to communicate between EXEC files at different recursion levels.

Thus, if CHECK EXEC contains:

```
&GLOBAL1 = 100
EXEC COUNTEM
```

The variable &GLOBAL1 has a value of 100 in COUNTEM EXEC, which may also test and modify it.

Horizontal communication by means of global variables is also possible at recursion levels 2 and above. For example: EXEC A calls EXEC B, which sets &GLOBAL1 to 2 and exits, then EXEC A (STILL ACTIVE) calls EXEC C, which finds that &GLOBAL1 has a value of 2, as set by EXEC B.

The CMS EXEC interpreter can handle up to 19 levels of recursion at one time, that is, up to 19 EXECs may be active, one nested within another. An EXEC may also call itself.

You can test the &GLOBAL special variable to see if an EXEC is executing within another procedure and if so, at what level of recursion it is executing. For example, if the file RECOMP EXEC contained the lines:

```
&IF &GLOBAL EQ 2 &GOTO -2NDPASS
.
.
.
EXEC RECOMP
.
.
.
-2NDPASS &TYPE SECOND PASS BEGINS
```

then when the line "EXEC RECOMP" is executed, control passes to the beginning of the EXEC; the value of &GLOBAL changes from 1 to 2; and control is passed to the &TYPE statement at the label 2NDPASS.

Exiting From CMS EXEC Procedures

Execution in a CMS EXEC procedure proceeds sequentially through a file, line by line. When a statement causes control to be passed to another statement, execution continues at the second statement, and again proceeds sequentially through the file. When the end of the file is reached, the EXEC terminates processing. Frequently, however, you may not want processing to continue to the end of the file. You can use the &EXIT statement to cause an immediate exit from EXEC processing and a return to the CMS environment. If the EXEC has been invoked from another EXEC, control is returned to the calling EXEC file. For example, the statement:

```
&IF &RETCODE ¬= 0 &EXIT
```

would cause an immediate exit from the EXEC if the return code from the last issued CMS command was not zero.

You can use the &EXIT statement to terminate each of a series of execution paths in an EXEC. For example, using the following statements,

```
&IF &1 EQ PRINT &GOTO -PRINT
&IF &1 EQ TYPE &GOTO -TYPE
.
.
.
-PRINT
.
.
.
&EXIT
-TYPE
.
.
.
&EXIT
```

you ensure that once the path through the -PRINT routine has been taken, the EXEC does not continue processing with the -TYPE routine.

Passing Return Codes From EXECs: The &EXIT control statement also provides a special function that allows you to pass a return code to CMS or to the program or EXEC that called this EXEC. You specify the return code value on the &EXIT control statement as follows:

```
&EXIT 4
```

Execution of this line results in a return to CMS with a ready message:

```
R(00004);
```

If you have a variety of exits in an EXEC, you can use a different value following each &EXIT statement, to indicate which path had been taken in the EXEC.

You can also use a variable symbol as the value to be passed as the return code:

```
&EXIT &VAL
```

Another common use of the &EXIT statement is to cause an exit to be taken following an error in a CMS command, and using the return code from the CMS command in the &EXIT statement:

```
&IF &RETCODE NE 0 &EXIT &RETCODE
```

Terminal Communications

You can design EXECs to be used interactively, so that their execution depends on information entered directly from the terminal during the execution. With the &TYPE statement, you can display a line at the terminal, and with the &READ statement, you can read one or more lines from the terminal or console stack. Used together, these statements can provide a prompting function in an EXEC:

```
&TYPE WHAT DO YOU WANT TO DO NOW?
&TYPE ENTER (STOP CONTINUE REPEAT):
&READ VARS &LABEL
&GOTO -&LABEL
-STOP
.
.
.
-CONTINUE
.
.
.
-REPEAT
.
.
.
```

In this example, the &READ control statement is used with the VARS operand, which accepts the words entered at the terminal as values to be assigned to variable symbols. If the word STOP is entered in response to the &READ VARS statement in this example, the variable symbol &LABEL is assigned the value STOP. Then, in the &GOTO statement, the symbol is substituted with the value STOP, so the branch is taken to the label -STOP.

You can specify up to 17 variable names on an &READ VARS control statement. If you enter fewer words than are expected, the remaining variables are set to blanks. If you enter a null line, any variable symbols on the &READ line are set to blanks. If the execution of your EXEC depends on a value entered as a result of an &READ VARS, you might want to include a test for a null line immediately following the statement; for example:

```
&READ VARS &TITLE &SUBJ
&IF .&TITLE = . &EXIT 100
```

If no tokens are entered in response to the terminal read request, the variable &TITLE is null, and the EXEC terminates with a return code of 100.

If you are writing an EXEC that may receive a number of tokens from the terminal, you may find it more convenient to use the &READ ARGS form of the &READ control statement. When the &READ ARGS statement reads a line from the terminal, the tokens entered are assigned to the &n special variables (&1, &2, and so on).

Reading CMS Commands and CMS EXEC Control Statements from the Terminal

When you use the &READ control statement with no operands, or with a numeric value, EXEC reads one line or the specified number of lines from the terminal. These lines are treated, by EXEC, as if they were in the EXEC file all along. For example, if you have an EXEC named COMMAND that looks like the following:

```
&TYPE ENTER NEXT COMMAND:
&READ 1
&SKIP -2
```

all the commands you enter during the terminal session are processed by the EXEC. Each time the &READ statement is executed, you enter a CMS command. When the command finishes, control returns to EXEC, which prompts you to enter the next command. Since the CMS commands are all being executed from within the EXEC, you do not receive the CMS ready message at the completion of each command.

You could also enter EXEC control statements or assignment statements. To terminate the EXEC and return to the CMS environment, you must enter the EXEC control statement &EXIT from the terminal:

```
&exit
```

Displaying Data at a Terminal

You can use the &TYPE and &BEGTYPE control statements to display lines from your EXEC at the terminal. In addition, you can use the CMS TYPE command to display the contents of CMS files. When you use the &TYPE control statement, you can display variable symbols as well as data. Variable symbols on an &TYPE control statement are substituted before they are displayed. For example, the lines:

```
&NAME = ARCHER
&TYPE &NAME
```

result in the display:

```
ARCHER
```

You can use the &TYPE statement to display prompting messages, error or information messages, or lines of data.

In an EXEC file with fixed-length records, only the first 72 characters of each line are processed by the EXEC interpreter. Therefore, if you want to use the &TYPE control statement to display a line longer than 72 characters, you must convert the file into variable-length records.

&BEGTYPE and &BEGTYPE ALL

All of the words in an &TYPE control statement are scanned into 8-character tokens. If you need to display a word that has more than 8 characters, you

must use the &BEGTYPE control statement. The &BEGTYPE control statement precedes one or more data lines that you want to display; for example:

```
&BEGTYPE
THIS EXEC PERFORMS THE FOLLOWING FUNCTIONS:
1. IT ACCESSES DISKS 193, 194, and 195 AS
   B, C, AND D EXTENSIONS OF THE A-DISK.
2. IT DEFINES, FORMATS, AND ACCESSES A
   TEMPORARY 195 DISK (E).
&END
```

The &END statement must be used to terminate a series of lines introduced with the &BEGTYPE statement. "&END" must begin in column 1 of the EXEC file.

The lines following an &BEGTYPE statement, up to the &END statement, are not scanned by the EXEC interpreter. Therefore, no substitution is performed on the variable symbols on these data lines. If you need to display a symbol, you must use the &TYPE control statement. To display a combination of scanned and unscanned lines, you might need to use both the &TYPE and &BEGTYPE control statements:

```
&BEGTYPE
EVALUATION BEGINS...
&END
&TYPE &VAL1 IS &NUM1.
&TYPE &VAL2 IS &NUM2.
&BEGTYPE
EVALUATION COMPLETE.
&END
```

If you use the &BEGTYPE control statement in an EXEC file with fixed-length records, and you want to display lines longer than 72 characters, you must use the ALL operand. For example:

```
&BEGTYPE ALL
 ..&perioddata line of 103 characters
 ..&perioddata line of 98 characters
 ..&perioddata line of 50 characters
&END
```

You can display lines of up to 130 characters in this way. When you enter lines that are longer than the record length in an EXEC file, the records are truncated by the editor. You must increase the record length of the file by using the LRECL option of the EDIT command, for example:

```
edit old exec a (lrecl 130
```

Using the CMS TYPE Command

You can use the TYPE command in an EXEC file to display data files, or portions of data files. For example, you might have a number of files with the same filetype; the files contain various kinds of data. You could create an EXEC that invokes the TYPE command to display a particular file as follows:

```
&IF &INDEX EQ 2 &IF &2 EQ ? &GOTO -TYPE
.
.
.
-TYPE
ACCESS 198 B
TYPE &1 MEMO B
```

The filetype MEMO is a reserved filetype, which accepts data in uppercase and lowercase; you can use it for documentation files or programming notes.

Controlling Terminal Displays

The two CMS Immediate commands that control terminal display are HT (halt typing) and RT (resume typing). When data is being displayed at your terminal, you can suppress the display by signaling an attention interruption and entering:

```
ht
```

This command affects output that is being displayed:

- As a response to a CMS command, including prompting messages, error messages, or normal display responses (as from the TYPE command)
- From a program
- In response to an &TYPE or &BEGTYPE request in an EXEC

Once display has been suppressed, and before the command, program, or EXEC completes execution, you can request that display be resumed by signaling another interruption and entering:

```
rt
```

In an EXEC file, if you want to halt or resume display, you must use the &STACK control statement to enter the RT or HT commands or use the CMS commands SET CMSTYPE RT and SET CMSTYPE HT. For example, the ACCESS command issues a message when a disk is accessed:

```
D(198) R/O
```

If you are going to issue the ACCESS command within a CMS EXEC and you do not wish this message displayed, you could enter the lines:

```
SET CMSTYPE HT
ACCESS 198 D
  or
&STACK HT
ACCESS 198 D
```

When you halt CMS terminal display with an HT command, all displaying, except for CMS Error messages with a suffix letter of S or T, is suppressed for the remainder of the EXEC file's execution. To reverse the suppressed display, use the RT immediate command or the SET CMSTYPE RT. To execute the RT Immediate command in an EXEC, use either of the statements:

```
&STACK RT
 -- or --
SET CMSTYPE RT
```

The &TYPEFLAG Special Variable: You can test the current value of the display controlling an EXEC with the &TYPEFLAG special variable. The value of &TYPEFLAG can only be one of the literal values HT or RT. For example:

```
&IF &$ EQ NOTYPE &STACK HT
.
.
.
&IF &TYPEFLAG EQ HT &SKIP 3
&TYPE LINE1
&TYPE LINE2
&TYPE LINE3
&CONTINUE
```

In this example, if NOTYPE is entered as an argument when the EXEC is invoked, the CMS command SET CMSTYPE HT is executed, hence no display appears at the terminal. Within the EXEC, the variable &TYPEFLAG is tested, and, if it is HT, then a series of &TYPE statements is skipped. Since EXEC does not have to process these lines, you can save time and system resources by not processing them.

Reading from the Console Stack

When you are in the CMS environment executing programs or CMS commands, you can stack commands, either by entering multiple command lines separated by the logical line end symbol, as follows:

```
print myfile listing#cp query printer
```

or by signaling an attention interruption and entering a command line, as follows:

```
print myfile listing
!
cp query printer
```

In both of the preceding examples, the second command line is saved in the console stack. Whenever a read occurs in your virtual machine, CMS reads lines from the console stack, if there are any lines in it. If there are no lines in the stack, the read results in a physical read to your terminal (on a typewriter terminal, the keyboard unlocks).

A virtual machine read occurs whenever a command or subcommand finishes execution, or when an EXEC or a program issues a read request. Many CMS commands also issue read requests, for example, SORT and COPYFILE. If you want to execute one of these commands in an EXEC, you may want to stack, in the console stack, the response to the read request so that when it is issued it is immediately satisfied. For example:

```
&STACK 42-121 1
COPYFILE &NAME LISTING A = ASSEMBLE = (SPECS
```

When the COPYFILE command is issued with the SPECS option, a prompting message for a specification list is issued, followed by a read request. In this EXEC, the request is satisfied with the line stacked with the &STACK control statement. If the response were not stacked, you would have to enter the appropriate information from the terminal during the execution of the EXEC that contained this COPYFILE command line.

In addition to stacking predefined responses to commands and programs, you can use the console stack to stack CMS commands and EDIT subcommands, as well as data lines to be read within the EXEC.

The number of lines that you can place in the console stack at any one time varies according to the amount of storage available in your virtual machine for stacking.

You may want to stack one or two lines at a time, or you may wish to stack many lines. There are several features available in EXEC that can help you manipulate the stack.

Exchanging Data Between Programs through the Stack

The console stack is composed of the terminal input buffer and the program stack. Lines typed at the terminal (maximum length of 130 characters per line) are placed in the terminal input buffer. Lines transmitted by programs through the CMS ATTN function are placed in the program stack (maximum length of 255 characters per line).

When the WAITRD function is called (as a result of a RDTERM macro call, for example), it will look in the most recently created buffer of the program stack (see BUFFER #2 in Figure B-5). As each buffer is exhausted, RDTERM will look to the next buffer in the program stack (BUFFER #1). If the program stack is empty, WAITRD will then look in the terminal input buffer for an input line. If the terminal input buffer is also empty, then a "console read I/O" will be issued to acquire data from the terminal.

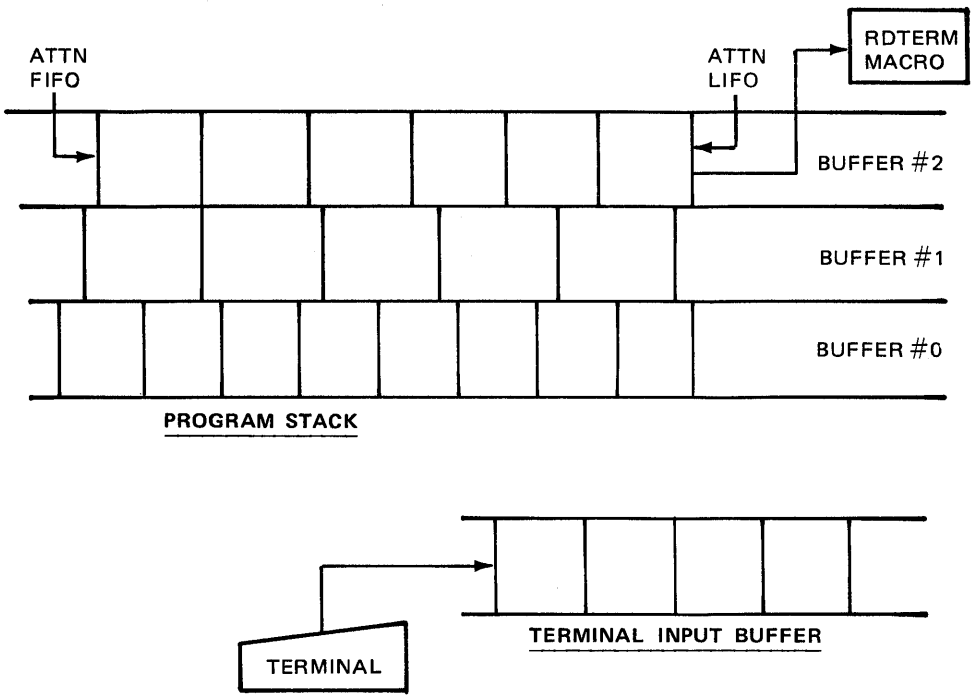


Figure B-5. The Console Stack

However, when a program issues a RDTERM TYPE=DIRECT, a VM READ is presented at your terminal. The program stack and terminal input buffer are bypassed and unchanged. When the response is entered, the first "logical line" is read and transferred to buffer. If multiple "logical lines" are entered, the remaining lines are added to the terminal input buffer in a FIFO manner.

Previously stacked lines read from the program stack will not have changed since the time they were stored by ATTN (unless uppercase translation has been requested). Before lines are extracted from the terminal input buffer, they are scanned by CP (when typed) for characters defined by the CP TERMINAL

command (or for their default values). WAITRD will then scan them for X'15' (logical end of line character), X'00' (physical end of line character), and for any other character defined through a CMS 'SET INPUT' command.

The MAKEBUF, DROPBUF, SENTRIES, and DESBUF CMS commands allow you to create buffers in the program stack, eliminate some or all of the program stack buffers, determine the number of lines in the program stack, and empty both the program stack and the terminal input buffer. These commands may also be called from a terminal (as CMS commands), from EXEC files, or from assembler language programs. A complete description of these commands can be found in the publication *VM/SP CMS Command and Macro Reference.*

Note: Lines read from the terminal or stacked in the terminal input buffer can be restacked in the program stack, using the ATTN function, and executed at a later time. The line length specified in the parameter list for the ATTN function should be the same length as the line that was previously read from the terminal or the terminal input buffer. A line stacked again by ATTN, using a line length greater than the line length read from the terminal or the console input buffer, may result in an error when execution of the stacked line is attempted.

&BEGSTACK and &BEGSTACK ALL

Just as the &TYPE control statement has an &BEGTYPE counterpart, the &STACK control statement has an &BEGSTACK counterpart. You can stack multiple data lines following an &BEGSTACK statement. Lines stacked in this way are not scanned by the EXEC processor, and no substitution is performed on variable symbols. For example, the lines:

```
&BEGSTACK
..&periodline of data
..&periodline of data
..&periodline of data
&END
```

stack three data lines in the stack. The stacked lines must be followed by an &END control statement, which must be entered in the EXEC file beginning in column 1.

If you have an EXEC with fixed-length records, and you want to stack data lines longer than 72 characters, you must use the ALL operand of the &BEGSTACK control statement:

```
&BEGSTACK ALL
..&periodline of 103 characters
..&periodline of 98 characters
..&periodline of 60 characters
&END
```

The ALL operand is not necessary for variable-length EXEC files.

Stacking FIFO and LIFO

When you are stacking multiple lines in an EXEC, you may choose to reverse the sequence in which lines are read in from the stack. The default sequence is FIFO (first-in, first-out), but you may specify LIFO (last-in, first-out) when you enter the &STACK or &BEGSTACK control statement. For example, execution of the lines:

```
&STACK &TYPE A
&STACK &TYPE B
&STACK LIFO &TYPE C
&STACK LIFO &TYPE D
&STACK &TYPE E
```

results in the display:

```
D
C
A
B
E
```

The &READFLAG Special Variable

The EXEC special variable &READFLAG always contains one of two values, STACK or CONSOLE. When it contains the value STACK, it indicates that there are lines in the stack. When it contains the value CONSOLE, it indicates that the stack is empty and that the next read request results in a physical read to the terminal (console).

You can test this value in an EXEC, for example:

```
&IF &READFLAG EQ STACK &SKIP 2
&TYPE STACK EMPTY
&EXIT
&CONTINUE
```

You might use a similar test in an EXEC that processes a number of lines from the stack, and loops through a series of steps until the stack is empty.

Stacking CMS Commands

Whenever you place a command in the console stack, it remains there until a read request is presented to the terminal. If the request is the result of an &READ control statement, then the line is read from the stack. For example, the lines:

```
&STACK CP QUERY TIME
&READ
```

result in the command line being stacked, read in, and then executed.

If there are no read requests in an EXEC file, then any commands that are stacked are executed after the EXEC has finished and has returned control to the CMS environment. For example, consider the lines:

```
TYPE &1 LISTING A
ACCESS 198 A
TYPE &1 LISTING A
```

If this EXEC is located on your 191 A-disk, then when the ACCESS command accesses a new A-disk, CMS cannot continue reading the EXEC file and issues an error message. However, if the EXEC was written as follows:

```
TYPE &1 LISTING A
&STACK ACCESS 198 A
&STACK TYPE &1 LISTING A
```

then, after the TYPE command, the next command lines are stacked, the EXEC finishes executing and returns control to CMS, which reads the next command lines from the console stack.

When you stack either CMS commands or data with the &STACK control statement in an EXEC procedure, the EXEC processor treats everything except the immediate commands, HT and RT, as data. Control characters such as the character delete and line end character are not recognized and therefore not interpreted as performing any special function. The logical control characters as defined in the CP TERMINAL command, are not substituted with their special values, since the EXEC lines are being read from disk and not from a terminal.

Stacking EDIT Subcommands

If you want to issue the EDIT command from within an EXEC, you might want to stack EDIT subcommands to be read by the CMS editor. You should stack these subcommands, either with &STACK statements, or with the &BEGSTACK statement, just before issuing the EDIT command. For example:

```
&BEGSTACK
CASE M
GET SETUP FILE A 1 20
TOP
LOCATE /XX/
&END
&STACK REPLACE
EDIT &1 DATA (LRECL 120
```

If this EXEC is named EDEX, and you invoke it with:

```
edex fr01
```

the EDIT subcommands are stacked in the order they appear in the EXEC. The EDIT command is invoked to edit the file FR01 DATA, and the EDIT subcommands are read from the stack and executed. When the stack is empty, your virtual machine is in the edit environment in input mode, and the first line you enter replaces the existing line that contains the character string XX.

Note: All of the EDIT subcommands in the example, except for the REPLACE subcommand, are stacked within an &BEGSTACK stack, and that the REPLACE subcommand is stacked with &STACK. If you are creating EXEC files with fixed-length records, you must use &STACK to stack the INPUT and REPLACE subcommands. If you use &BEGSTACK, then the INPUT and REPLACE subcommands are treated as if they contain text data, and so insert or replace one line in the file (a line of blanks). This is not true, however, for variable-length EXEC files.

Similarly, if you want to stack a null line, to change from input mode to edit mode in an EXEC, you must use the &STACK statement with no other data on the line (in both fixed- and variable-length EXEC files). For example:

```
&STACK INPUT
&BEGSTACK
 ..&perioddata line
 ..&perioddata line
 ..&perioddata line
&END
&STACK
&STACK FILE
EDIT &1 &2
&EXIT
```

When this EXEC is invoked with a filename and filetype as arguments, the INPUT subcommand, data lines, null line, and FILE subcommand are placed in the stack before the EDIT command is issued. The data lines are placed in the specified file and the file is written onto disk before the EXEC returns control to CMS.

Stacking Lines for EXEC to Read

Lines in the console stack can be read by the EXEC interpreter with an &READ control statement. For example:

```
-SETUP
   &LOOP 3 &NUM = 50
   &STACK &NUM &CHAR
   &NUM = &NUM + 1
   &CHAR = &CONCAT &STRNG &NUM
.
.
.
-READ
   &LOOP -FINIS &READFLAG EQ CONSOLE
   &READ ARGS
.
.
.
  -FINIS
```

In this EXEC procedure, the statements following the label -SETUP stack a number of lines. The variables &NUM and &CHAR are substituted before they are stacked. At the label -READ, the lines are read in from the stack and processed. The values stacked are read in as the variable symbols &1 and &2. Control passes out of the loop when the stack is empty.

Clearing the Console Stack

If you use the console stack in an EXEC procedure, you should be sure that it is empty before you begin stacking lines in it. Also, you should be sure that it is empty before exiting from the EXEC (unless you have purposely stacked CMS commands for execution).

One way to clear a line from the stack without affecting the execution of your EXEC is to use the &READ VARS or &READ ARGS control statement. You can use &READ VARS without specifying any variable symbols so that the line read is read in and effectively ignored. For example:

```
&LOOP 1 &READFLAG EQ CONSOLE
&READ ARGS
```

If these lines occur at the beginning of an EXEC file, they ensure that any stacked lines are cleared. If the EXEC is named EXEC1 and is invoked with the line:

```
exec1#type help memo#type print memo
```

then the lines TYPE HELP MEMO and TYPE PRINT MEMO are cleared from the stack and are not executed.

You could use the same technique to clear the stack in case of an error encountered in your EXEC, so that the stack is cleared before returning to CMS. You would especially want to do this if you stacked data lines or EXEC control statements that have no meaning to CMS.

Another way to clear the console stack is with the CMS function DESBUF. For example:

```
&IF &READFLAG EQ STACK DESBUF
```

When you use the DESBUF function to clear the console input stack, the output stack is also cleared. The output stack contains lines that are waiting to be displayed or typed at the terminal. Frequently, when an EXEC is processing, output lines are stacked, and are not displayed immediately following the execution of an &TYPE statement. If you want to display all pending output lines before clearing the console input stack, you should use the CONWAIT function, as follows:

```
CONWAIT
&IF &READFLAG EQ STACK DESBUF
```

The CONWAIT function causes a suspension of program execution until the console output stack is empty. If there are no lines waiting to be displayed, CONWAIT has no effect.

Clearing the stack is important when you write edit macros, since all subcommands issued in an edit macro must be first stacked. See "Writing Edit Macros" for additional information on using the console stack.

File Manipulation with CMS EXECs

You can, to a limited degree, read and write CMS disk files using EXECs. You can stack files with a filetype of EXEC in the console stack and then read them, one record at a time, with &READ control statements. All data items are truncated to eight characters. You can write a file, one record at a time, with the &PUNCH control statement, and then you can read the spool punch file onto disk. Examples of these techniques follow.

Stacking EXEC Files

There are two methods to stack EXEC files in the console stack. One is illustrated using a CMS EXEC file, as shown in the following PREFIX EXEC:

```
&LNAME = &CONCAT &1 *
LISTFILE &LNAME SCRIPT * (EXEC
EXEC CMS &STACK
&LOOP -END &READFLAG EQ CONSOLE
&READ VARS &NAME &TYPE &MOD
&SUFFIX = &SUBSTR &NAME 3 6
&NEWNAM = &CONCAT &2 &SUFFIX
RENAME &NAME &TYPE &MOD &NEWNAM &TYPE &MOD
&IF &RETCODE EQ 0 &SKIP
&TYPE FILE &NAME &TYPE NOT RENAMED
-END
```

This EXEC procedure is invoked with two arguments, each two characters in length, which indicate old and new prefixes for filenames. The EXEC renames files with a filetype of SCRIPT that have the first prefix, changing only the prefix in the filename.

The LISTFILE command, invoked with the EXEC option, creates a CMS EXEC file in the format:

```
&1 &2 filename SCRIPT mode
```

When the EXEC is invoked with the line:

```
EXEC CMS &STACK
```

the argument &STACK is substituted for the variable symbol &1 in each line in the CMS EXEC. The execution of the CMS EXEC effectively stacks, in the console stack, the complete file identifications of the files listed:

```
&STACK filename SCRIPT mode
&STACK filename SCRIPT mode
.
.
.
```

These stacked lines are read back into the EXEC, one at a time, and the tokens "filename," "SCRIPT," and "mode" are substituted for the variable symbols &NAME, &TYPE, and &MOD.

Using the &SUBSTR and &CONCAT built-in functions, the new name for each file is constructed, and the RENAME command is issued to rename the files.

For example, if you invoke the EXEC procedure with the line:

```
prefix ab xy
```

all SCRIPT files that have filenames beginning with the characters AB are renamed so that the first two characters of the filename are XY. A sample execution summary of this EXEC is illustrated under "Debugging EXEC Procedures."

Stacking Data Files

You can create a data file, containing fixed-length records, using a filetype of EXEC. To stack these data lines in the console stack, you can enter them following an &BEGSTACK (or &BEGSTACK ALL) control statement. For example, the file DATA EXEC is as follows:

```
&BEGSTACK
HARRY 10/12/48
PATTI 1/18/49
DAVID 5/20/70
KATHY 8/6/43
MARVIN 2/28/50
```

The file BDAY EXEC contains:

```
&CONTROL ERROR
EXEC DATA
&IF &READFLAG EQ CONSOLE &GOTO -NO
&READ VARS &NAME &DATE
&IF &NAME NE &1 &SKIP -2
-FOUND
&IF .&1 EQ . &EXIT
&TYPE &1 'S BIRTHDAY IS &DATE
CONWAIT
DESBUF
&EXIT
-NO &TYPE &1 NOT IN LIST
&EXIT
```

When the BDAY EXEC is invoked, it expects an argument that is a first name. The function of the EXEC is to display the birthday of the specified person. A sample execution of this EXEC might be:

```
bday kathy
KATHY 'S BIRTHDAY IS 8/6/43
R;
```

BDAY EXEC first executes the DATA EXEC, which stacks names and dates in the console stack. Then, BDAY EXEC reads one line at a time from the stack, assigning the variable names &NAME and &DATE to the tokens on each line. It compares &NAME with the argument read as &1. When it finds a match, it displays the message indicating the date, and clears the console stack after waiting for terminal output to finish.

Note: The file DATA EXEC begins with an &BEGSTACK control statement, but contains no &END statement. The stack is terminated by the end of the EXEC file. "Writing Data Files" describes a technique you might use to add new names and birth dates to the DATA EXEC file.

Writing Data Files

You can build a CMS file in your virtual card punch using the &PUNCH and &BEGPUNCH control statements. Depending on the spooling characteristics of your virtual punch, the file that you build may be sent to another user's card reader, or to your own virtual card reader. When you read the file with the CMS READCARD command, the spool reader file becomes a CMS disk file.

The following example illustrates how you might use your card punch and reader to update a CMS file by adding records to the end of it. The file being updated is the DATA EXEC, which is the input file for the BDAY EXEC, shown in the example in "Stacking Data Files." You could create a separate CMS EXEC to perform the update, but this example shows how you might modify the BDAY EXEC to perform the addition function (ellipses indicate the body of the EXEC, which is unchanged):

```
&CONTROL ERROR
&IF &1 EQ ADD &GOTO -ADDNAME
.
.
.
&EXIT
-ADDNAME
&TYPE ENTER FIRST NAME AND DATE IN FORM MM/DD/YY
&READ VARS &NAME &DATE
&IF .&NAME = . &SKIP 3
&PUNCH &NAME &DATE
&TYPE ENTER NEXT NAME OR NULL LINE:
&SKIP -4
CP SPOOL PUNCH TO *
CP CLOSE PUNCH
READCARD NEW NAMES
COPYFILE NEW NAMES A DATA EXEC A (APPEND
&IF &RETCODE = 0 &SKIP 2
&TYPE ERROR CREATING FILE
&EXIT &RETCODE
ERASE NEW NAMES
```

When BDAY EXEC is invoked with the keyword ADD, you are prompted to enter lines to be added to the data file. Each line that you enter is punched to the card punch. When you enter a null line, indicating that you have finished entering lines, the CP commands SPOOL and CLOSE direct the spool file to your card reader and close the punch.

The file is read in as the file NEW NAMES, which is appended to the file DATA EXEC using the COPYFILE command with the APPEND option. The file NEW NAMES is erased and the EXEC terminates processing.

Using Your Virtual Card Punch

When you punch lines in your virtual punch, the lines are not released as a CP spool file until the punch is closed. Since the EXEC processor does not close the virtual punch when it terminates processing, you must issue the CLOSE command to release the file. You can do this in the EXEC with the command line:

```
CP CLOSE PUNCH
```

or from the CMS environment after the EXEC has finished. If you use the CLOSE command in the EXEC, you must preface it with CP.

The CMS PUNCH command, which you can use in a CMS EXEC to punch an entire CMS file, closes the punch after punching a file. Therefore, if you want to create a punch file using a combination of &PUNCH control statements and PUNCH commands, you must spool your punch using the CONT option, so that a close request does not affect the file:

```
CP SPOOL PUNCH TO * CONT
&PUNCH FIRST FILE
&PUNCH
PUNCH FILE1 TEST ( NOHEADER
&PUNCH SECOND FILE
&PUNCH
PUNCH FILE2 TEST ( NOHEADER
CP SPOOL PUNCH CLOSE NOCONT
```

The preceding example punches title lines introducing the files punched with the CMS PUNCH command. The null &PUNCH statements punch blank lines. The PUNCH command is issued with the NOHEADER option, so that a read control card is not punched.

You can also use an EXEC procedure to punch a job to send to the CMS batch facility for processing. The batch facility, and an example of using an EXEC to punch a job to it, are described in Chapter 12, "Using the CMS Batch Facility"

Using &PUNCH and &BEGPUNCH

All lines punched to the virtual card punch are fixed-length, 80-character records. When you use the &PUNCH control statement in a fixed-length EXEC file, EXEC scans only the first 72 columns of the EXEC.

If you want to punch a word that contains more than eight characters, you must use the &BEGPUNCH control statement, which also, in fixed-length files, causes EXEC to punch data in columns 1 through 80.

Using CMS EXECs with CMS Commands

Whenever you create a CMS EXEC file you are, for all practical purposes, creating a new CMS command. When you enter a command line in the CMS environment, CMS searches for a CMS EXEC file with the specified filename before searching for a MODULE file or CMS command. You can place the names of your EXEC files in a synonym table and assign minimum truncation values for the synonyms, just as you can for CMS command names.

While many of your EXEC procedures may be very simple, others may be very long and complicated, and perform many of the housekeeping functions performed by CMS commands, such as syntax checking, error message generation, and so on.

Monitoring CMS Command Execution

Many, or most, of your EXEC procedures may contain sequences of CMS commands that you want to execute. If your EXEC procedure contains no EXEC control statements, each command line is displayed and then the command is executed. If an error occurred, the CMS error message is displayed, followed by a return code in the format:

```
+++ R(nnnnn) +++
```

where nnnnn is the nonzero return code from the CMS command. If the command is not a valid CMS command, or the command function for SET or QUERY is invalid and the implicit CP function is in effect, the return code is a -3:

```
+++ R(-0003) +++
```

You may also receive this error return when you use a CP command without prefacing it with the CP command. If you enter an unknown CP command following "CP," you receive a return code of 1.

If a command completes successfully, no return code is displayed.

If you do not want to see the command lines displayed before execution, nor return codes following execution, you can use the EXEC control statement:

```
&CONTROL OFF
```

Or, if you want to see only the command lines that produced errors, and the resultant return codes, you can specify:

```
&CONTROL ERROR
```

Regardless of these settings of the &CONTROL statement, CMS error messages are displayed, as long as the value of &READFLAG is RT, and the terminal is displaying output.

If you issue the LISTFILE, STATE, ERASE, or RENAME commands in an EXEC procedure, and you do not want to see the error message FILE NOT FOUND displayed, you can use the statement:

```
&CONTROL NOMSG
```

to suppress the display of these particular messages.

You can request that particular timing information be displayed during an EXEC's execution. If you want to display the time of day at which each command executes, you can specify:

```
&CONTROL TIME
```

Then, as each command line is displayed, it is prefaced with the time; for example:

```
&CONTROL CMS TIME
QUERY BLIP
```

executes as follows:

```
10:34:16 QUERY BLIP
BLIP     = *
```

If you wish to see, following the execution of each CMS command, specific CPU timing information, such as the long form of the ready message, you can use the &TIME control statement. For example:

```
&TIME ON
QUERY BLIP
QUERY FILEDEF
```

might execute as:

```
QUERY BLIP
BLIP     = OFF
T=0.01/0.04 10:44:21

QUERY FILEDEF
NO USER DEFINED FILEDEF'S IN EFFECT
T=0.01/0.04 10:45:26
```

Handling Error Returns From CMS Commands

In many cases, you want to execute a command only if previous commands were successful. For example, you would not want to execute a PRINT command to print a file if you had been unable to access the disk on which the file resided. There are two methods, using EXEC procedures, that allow you to monitor and control what happens following the execution of CMS commands. One method uses the EXEC control statement &ERROR to transfer control when an error occurs; the other tests the special variable &RETCODE upon completion of a CMS command to determine whether that particular command completed successfully.

Using the &ERROR Control Statement

When a CMS command is executed within a CMS EXEC, a return code is passed to the EXEC interpreter, indicating whether or not the command completed successfully. If the return code is nonzero, EXEC then activates the &ERROR control statement currently in effect. For example, if the following statement is included at the beginning of an EXEC file:

```
&ERROR &EXIT
```

then, whenever a CMS command (or user program) completes with a nonzero return code, the &EXIT statement in the &ERROR statement is executed, and the EXEC terminates processing. You might use a similar statement in your EXECs to ensure that they do not attempt to continue processing in the event of an error.

An &ERROR control statement can specify any executable statement. It may transfer control to another portion of the EXEC, or it may be a single statement that executes before control is returned to the next statement in the EXEC. For example:

```
&ERROR &GOTO -EXIT
```

transfers control to the label -EXIT, in case of any CMS error. The statement:

```
&ERROR &TYPE CMS ERROR
```

results in the display of the message "CMS ERROR" before returning control to the statement following the command that caused the error.

If you do not have an &ERROR control statement in an EXEC, or if you specify &ERROR with no operands, EXEC takes no special action when a CMS command returns with an error return code. Specifying &ERROR with no operands is the same as specifying:

```
&ERROR &CONTINUE
```

Since an &ERROR control statement remains in effect for the remainder of the EXEC execution, or until another &ERROR control statement is encountered, you may use &ERROR &CONTINUE to restore default processing.

Using the &RETCODE Special Variable

An error return from a CMS command, in addition to calling an &ERROR control statement, also places the return code value in the EXEC special variable &RETCODE. Following the execution of any CMS command in an EXEC procedure, you can test whether or not the command completed without error. For example:

```
TYPE ALPHA FILE A
&IF &RETCODE ¬= 0 &EXIT
TYPE BETA FILE A
&IF &RETCODE ¬= 0 &EXIT
```

Note: The value of &RETCODE is modified after the execution of each CMS command.

The value of &RETCODE is affected by your own programs. If you execute a program in your EXEC using the LOAD and START (or FETCH and START) commands, or if you execute a MODULE file, then the &RETCODE special variable contains whatever value was in general register 15 when the program exited. If you are nesting EXEC procedures, then &RETCODE contains the value passed from the &EXIT statement of the nested EXEC.

You can use the value of the return code, as well, to analyze the extent or the cause of the error and to set up an error analysis routine accordingly. For example, suppose you want to set up an analysis routine to identify return codes 1 through 11 and to exit from the EXEC when the return code is greater than 11. When a return code is identified, control is passed to a corresponding routine that attempts to correct the error. You could set up such an analysis routine as follows:

```
-ERRANAL
&CNT = 0
&LOOP 2 &CNT EQ 12
&IF  &RETCODE EQ &CNT &GOTO -FIX&CNT
&CNT = &CNT + 1
.
.
.
-FIX0 &GOTO -ALLOK
-FIX1
.
.
.
&GOTO -ALLOK
-FIX2
.
.
.
&GOTO -ALLOK
.
.
.
-FIX11
.
.
.
-ALLOK
```

When the value of the &CNT variable equals the return code value in &RETCODE, the branch to the corresponding -FIX routine is taken. Each corrective routine performs different actions, depending on its code, and finishes at the routine labeled -ALLOK.

You can, in some cases, determine the cause of a CMS command error and attempt to correct it in your EXEC. To do this, you must know the return codes issued by VM/SP commands. See *VM/SP System Messages and Codes* for a discussion of the return codes for VM/SP commands. In addition, the error messages and corresponding return codes are listed under the command descriptions for each CMS command in the *VM/SP CMS Command and Macro Reference.*

As an example, all CMS commands that search for files issue a return code of 28 when a file is not found. If you want to test for a file-not-found condition in your EXEC, you might use statements similar to the following:

```
&CONTROL OFF NOMSG
.
.
.
TYPE HELP MEMO A
&IF &RETCODE = 28 &GOTO -NOFILE
```

Tailoring CMS Commands for Your Own Use

You can create CMS EXEC procedures that simplify or extend the use of a particular CMS command. Depending on your applications, you can modify the CMS command language to suit your needs. You can create EXEC files that have the same names as CMS commands, and, since CMS locates EXEC files before MODULE files, the EXEC is found first. For example, the COPYFILE command, when used to copy CMS disk files, requires six operands. If you change only the filename when you copy files, you could create a COPY EXEC as follows:

```
&CONTROL OFF
&IF &INDEX ¬= 3 &SKIP 2
COPYFILE &1 &2 = &3 &2 =
&EXIT
COPYFILE &1 &2 &3 &4 &5 &6 &7 &8 &9 &10 &11 &12 &13 &14 &15
```

If you always invoke the COPYFILE command using the truncation COPY, EXEC processes the command line for you, and if you have entered the three arguments, EXEC formats the COPYFILE command for you. If any other number of arguments is entered, the COPYFILE command is invoked with all the arguments as entered.

Creating Your Own Default Filetypes

If you use special filetypes for particular applications and they are not among those that the CMS editor supplies default settings for, but do require special editor settings, you can create a CMS EXEC to invoke the CMS editor. The CMS EXEC can check for particular filetypes, and if it finds them, stack the appropriate EDIT subcommands. If you name this EXEC procedure E EXEC, then you can bypass it by using a longer form of the EDIT command. The following is a sample E EXEC:

```
&CONTROL OFF
&IF &INDEX GT 1 &SKIP 2
EDIT &1 SCRIPT
&EXIT
&IF &2 EQ TABLE &GOTO -TABLE
&IF &2 EQ CHART &GOTO -CHART
&IF &2 EQ EXEC &GOTO -EX
&IF &2 EQ SYSIN &GOTO -SYSIN
-NORM EDIT &1 &2 &3 &4 &5 &6
&EXIT
-TABLE &BEGSTACK
IMAGE ON
TABS 1 10 20
CASE M
&END
EDIT &1 &2 &3 (LRECL 20
&EXIT
-CHART &BEGSTACK
CASE M
IMAGE ON
&END
EDIT &1 &2 &3
&EXIT
-EX
EDIT &1 &2 &3 (LRECL 130
&EXIT
-SYSIN &BEGSTACK
TABS 1 10 16 31 36 41 46 69 72 80
SERIAL ON
TRUNC 71
VERIFY 72
&END
EDIT &1 &2 &3
&EXIT
```

This CMS EXEC defines special characteristics for filetypes CHART, TABLE, and SYSIN, and defaults an EXEC file to 130-character records. If only one argument is entered, it is assumed to be the filename of a SCRIPT file. Since the editor is invoked from within the EXEC, control returns to EXEC after you use the FILE or QUIT subcommands during the edit session. You must use the &EXIT control statement so that the EXEC does not continue processing, and execute the next EDIT command in the file.

Refining Your CMS EXEC Procedures

This section provides supplementary information for writing complex EXEC procedures. Although the EXEC interpreter resembles, in some aspects, a high-level programming language, you do not need to be a programmer to write EXECs. Some of the techniques suggested here, for example, on annotating and writing error messages, are common programming practices, which help make programs self-documenting and easier to read and to use.

Annotating CMS EXEC Procedures

Lines in a CMS EXEC file that begin with an asterisk (*) are commentary and are treated as comments by the EXEC interpreter. You can use * statements to annotate your EXECs. If you write EXECs frequently, you may find it convenient to include a standard comment at the beginning of each EXEC, indicating its function and the date it was written, for example:

```
* EXEC TO HELP CONVERT LISTING FILES
* INTO SCRIPT FILES
*                J. BEAN 10/18/75
```

You can also use single asterisks or null lines to provide spacing between lines in an EXEC file to make examining the file easier.

In an EXEC, you cannot place comments on the same line with an executable statement. If you want to annotate a particular statement or group of statements, you must place the comments either above or below the lines you are annotating.

A good practice to use, when writing EXECs, is to set them up to respond to a ? (question mark) entered as the sole argument. For example, an EXEC named FSORT might contain:

```
&CONTROL OFF
&IF &INDEX = 1 &IF &1 = ? &GOTO -TELL
.
.
.
-TELL &BEGTYPE
      CORRECT FORM IS ' FSORT USERID <VADDR> '

       PRINTS AN ALPHABETIC LISTING OF ALL FILES ON THE SPECIFIED
       USER'S DISK. IF A VIRTUAL ADDRESS (VADDR) IS NOT
       SPECIFIED, THE USER'S 191 IS THE DEFAULT.
&END
```

You may also wish to anticipate the situation in which a user might enter an EXEC name with no arguments for an EXEC that requires arguments:

```
&IF &INDEX = 0 &GOTO -HELP
&IF &INDEX = 1 &IF &1 = ? &GOTO -TELL
.
.
.
&EXIT
-HELP &BEGTYPE
      CORRECT FORM IS ' COPY OLDFN OLDFT NEWFN '
      TYPE  ' COPY ? ' FOR MORE INFO
&END
&EXIT
-TELL &BEGTYPE
      CORRECT FORM IS ' COPY OLDFN OLDFT NEWFN '
      USES COPYFILE COMMAND TO CHANGE ONLY THE FILENAME
&END
&EXIT
```

This type of annotating is especially useful if you share your disks or your CMS EXECs with other users.

Error Situations

It is good practice, when writing CMS EXECs, to anticipate error situations and to provide meaningful error or information messages to describe the error when it occurs. The following error situations, and suggestions for handling them, have already been discussed:

- Errors in invoking the EXEC, either with an improper number of arguments, or with invalid arguments. (See "Arguments" in "Building CMS EXEC Procedures.")
- Errors in CMS command processing that can be detected with an &ERROR control statement or with the &RETCODE special variable. (See "Handling Error Returns from CMS Commands")

Many different kinds of errors may also occur, in the processing of your CMS EXEC control statements. EXEC processing errors, such as an attempt to branch to a nonexistent label or an invalid syntax, are "unrecoverable" errors. These errors always terminate CMS EXEC processing and return your virtual machine to the CMS environment or to the calling EXEC procedure or program. The error messages produced by EXEC, and the associated return codes, are described in the *VM/SP System Messages and Codes* .

Writing Error Messages

One way to make your CMS EXECs more readable, especially if they are long EXECs, is to group all of your error messages in one place, probably at the end of the EXEC file. You may also wish to number your messages and associate the message number with a label number and a return code. For example:

```
&IF &CT > 100 &GOTO -ERR100
&IF &CT < 0 &GOTO -ERR200
.
.
.
&IF &RETCODE EQ 28 &GOTO -ERR300
.
.
.
-ERR100
&TYPE COUNT TOO HIGH
&EXIT 100
-ERR200
&TYPE COUNT TOO LOW
&EXIT 200
-ERR300
&TYPE &1 &2 NOT ON DISK 'C'.
&EXIT 300
```

Using the &EMSG Control Statement

There is a facility, available in the EXEC processor, which allows you to write error messages that use the standard VM/SP message format, with an identification code and message number, as well as message text. When you use the &EMSG or &BEGEMSG control statement, the EXEC interpreter scans the first token and checks to see if the seventh (and last character) is an I, E, or W, representing information, error, or warning messages, respectively. If so, then the message is displayed according to the CP EMSG setting (ON, OFF, CODE, or TEXT). For example, if you have the statement:

```
&EMSG ERROR1E BAD ARGUMENT ' &1 '
```

the ERROR1E is considered the code portion of the message and BAD ARGUMENT is the text. If you have issued the CP command:

```
cp set emsg text
```

when this &EMSG statement is executed it may display:

```
BAD ARGUMENT ' PRNIT '
```

where PRNIT is the argument that is invalid. When you use &EMSG (or &BEGEMSG, which allows you to display error messages of unscanned data), the code portion of the message is prefixed with the characters DMS, when displayed. For example:

```
&BEGEMSG
ERROR2E INCOMPATIBLE ARGUMENTS
&END
```

displays when the EMSG setting is ON:

```
DMSERROR2E INCOMPATIBLE ARGUMENTS
```

You should use the &BEGEMSG control statement when you want to display lines that have tokens longer than eight characters; however, no variable substitution is performed.

Debugging CMS EXEC Procedures

If you have difficulty getting an EXEC procedure to execute properly, or if you are modifying an existing EXEC and wish to test it, there are a couple of simple techniques that you can use that may save you time.

One is to place the &CONTROL ALL control statement at the top of your EXEC file. When &CONTROL ALL is in effect, all the EXEC control statements are displayed before they execute, as well as the CMS command lines. One of the advantages of using this method is that the line is displayed after it is scanned, so that you can see the results of symbol and variable substitution.

"Stacking CMS EXEC Files" described a PREFIX EXEC, which changes the prefixes of groups of files. If the EXEC had an &CONTROL ALL statement, it might execute as follows:

```
prefix pt ag
&CONTROL ALL
&LNAME = &CONCAT PT *
LISTFILE PT* SCRIPT * ( EXEC
EXEC CMS &STACK
&LOOP -END &READFLA EQ CONSOLE
LOOP UNTIL:      STACK   EQ      CONS
&READ VARS &NAME &TYPE &MOD
&SUFFIX = &SUBSTR PTA 3 6
&NEWNAM = &CONCAT AG A
RENAME PTA SCRIPT A1 AGA SCRIPT A1
&IF 0 EQ 0 &SKIP
&SKIP
LOOP UNTIL:      STACK   EQ      CONS
&READ VARS &NAME &TYPE &MOD
&SUFFIX = &SUBSTR PTB 3 6
&NEWNAM = &CONCAT AG B
RENAME PTB SCRIPT A1 AGB SCRIPT A1
&IF 0 EQ 0 &SKIP
&SKIP
LOOP UNTIL:      CONSOLE EQ      CONS
R;
```

You can see from this execution summary that the files named PTA SCRIPT and PTB SCRIPT are renamed to AGA SCRIPT and AGB SCRIPT. Notice that the &LOOP statement results in a special LOOP UNTIL statement in the execution summary, which indicates the condition under which the loop executes.

Using CMS Subset

When you are using the CMS editor to create or modify a CMS EXEC procedure, you can test the EXEC in the CMS subset environment, as long as the EXEC does not issue any CMS commands that are invalid in CMS subset.

Before entering CMS subset with the CMS subcommand, you must issue the SAVE subcommand to write the current version of the EXEC onto disk; then, in CMS subset, execute the EXEC. For example:

```
edit new exec
NEW FILE:
EDIT:
input
INPUT:
&a = &1 + &2 + &3
&type answer is &a

EDIT:
save
EDIT:
cms
CMS SUBSET
new 34 56 899
ANSWER IS 989
R;
return
EDIT:
quit
R;
```

If the EXEC does not execute properly, you can return to the edit environment using the RETURN command, modify the EXEC, reissue the SAVE and CMS subcommands, and attempt to execute the EXEC again.

Summary of CMS EXEC Interpreter Logic

The following information is provided for those who have an interest in how the CMS EXEC interpreter works. It may help you in debugging your EXEC procedures if you have some idea of how processing is done by EXEC. When an EXEC file is invoked for execution, the EXEC interpreter examines each statement and analyzes it, according to the following sequence:

1. If the first nonblank character of the line is an *, the line is counted and ignored.

2. Null lines, except as a reponse to an &READ statement, are also counted and ignored.

3. The line is scanned, and nonblank character strings are placed in tokens.

4. All EXEC special variables, and then all user variables, except for those that appear as the target of an assignment statement, are substituted.

5. All blank tokens (resulting from the substitution of undefined symbols) are discarded.

6. If the first nonblank character is a hyphen (-), indicating a label, the next token is considered the first token.

7. If the first logical token does not begin with an ampersand (&), the line is passed to CMS for execution. The return code from CMS is placed in the special variable &RETCODE.

8. If the first logical token begins with an ampersand (&) EXEC interprets the statement.

9. If a statement is syntactically invalid and can be made valid by adding a token of blanks at the end, EXEC adds blanks, for example:

```
&BLANK =
&TYPE
&LOOP 3 &X NE
```

All of the above are valid EXEC control statements.

10. EXEC executes the statement. If no error is encountered, control passes to the next logical statement. If an error is encountered, EXEC terminates processing.

Note: For information on the EXEC 2 interpreter, see *VM/SP EXEC 2 Reference.*

Writing CMS EDIT Macros

If you have a good knowledge of the CMS EXEC facilities and an understanding of the CMS editor, you may wish to write edit macros. An edit macro is simply an EXEC file that contains a sequence of EDIT subcommands. Edit macros should only be invoked from the edit environment. An edit macro may contain a simple sequence of EDIT subcommands, or its execution may be dependent on arguments you enter when you invoke it. This section provides information on creating edit macros, suggestions on how to manipulate the console stack, and some examples of macros that you can create and use.

Creating CMS Edit Macro Files

An edit macro must have a filename beginning with a dollar sign ($) and a filetype of EXEC. Rules for file format, scanning and token substitution are the same as for all other EXEC files. A macro file may contain:

- EDIT subcommands
- CMS EXEC control statements
- CMS commands that are valid in CMS subset

When you create an edit macro that accepts arguments, you should be sure to check the validity of the arguments, and issue appropriate error messages. If you are writing an edit macro to expect arguments, you must keep in mind that the macro command line is scanned, and that any data items you enter are padded or truncated into eight-character tokens. Tokens are always translated to uppercase letters.

You should annotate all of your macro files, and provide a response to a question mark (?) entered as the sole argument (as described under "Annotating CMS EXEC Procedures."

How CMS Edit Macros Work

Since an edit macro is a CMS EXEC file, it is actually executed by the CMS EXEC interpreter, and not by the CMS editor. The CMS EXEC interpreter can only execute EXEC control statements and CMS commands. The only way to issue an EDIT subcommand from an EXEC file is to stack the subcommand in the console stack, so that when the editor is invoked, or receives control, it reads the subcommand(s) from the console stack before accepting input lines from the terminal. For example:

```
&STACK CASE M
&STACK RECFM V
EDIT &1 CHART A1
```

When the EDIT command is invoked from this EXEC, the CMS editor reads the subcommands from the stack and executes them.

To execute these same subcommands from an edit macro file, you must use the same technique; that is, you must place the subcommands in the console stack, for example:

```
&BEGSTACK
CASE M
RECFM V
&END
&EXIT
```

If this were an EXEC file named $VARY, you might execute it from the edit environment as follows:

```
edit test file
NEW FILE.
EDIT:
$vary
```

Stacked subcommands are executed only when the CMS EXEC completes its execution, either by reaching the end of the file, or by processing an &EXIT statement.

When you stack EDIT subcommands, you can use the &STACK and &BEGSTACK control statements. If you are stacking a subcommand that uses a variable expression, you must use the &STACK control statement, rather than the &BEGSTACK control statement. The following EXEC, named $T, displays a variable number of lines and then restores the current line pointer to the position it was in when the EXEC was invoked:

```
&CONTROL OFF
&IF &INDEX EQ 0 &GOTO -ERR
&CHECK = &DATATYPE &1
&IF &CHECK NE NUM &GOTO -ERR
&STACK TYPE &1
&UP = &1 - 1
&STACK UP &UP
&EXIT
-ERR &TYPE CORRECT FORM IS < $T N >
&EXIT 1
```

This edit macro uses the built-in function &DATATYPE to check that a numeric operand is entered.

CMS commands in an edit macro are executed as they are read by the CMS EXEC interpreter, just as they would if the EXEC were invoked in the CMS environment. You could create a $TYPE edit macro, for example, that would allow you to display a file from the edit environment:

```
&CONTROL OFF
TYPE &1 &2 &3 &4 &5 &6 &7
```

Or you might create a $STATE EXEC that would verify the existence of another file:

```
&CONTROL OFF
STATE &1 &2 &3
```

In both of these examples, the macro file invokes the CMS command. Macros like these can eliminate having to enter CMS subset environment to execute one or two simple CMS commands. You must be careful, though, not to execute any CMS command that uses the storage occupied by the editor. Only commands that are valid in CMS subset are valid in an edit macro.

The Console Stack

When you write an edit macro, you want to be sure that there are no EDIT subcommands in the stack that could interfere with the execution of the subcommands stacked by the macro file. Your macro should check whether there are any lines in the stack, and if there are, it should clear them from the stack. For example, you might use the lines:

```
&IF &READFLAG EQ CONSOLE &SKIP 2
DESBUF
&TYPE STACKED LINES CLEARED BY &0
```

The message "STACKED LINES CLEARED BY macro name" is issued by the edit macros distributed with the VM/SP system. $0 is the name of the macro. You may also want to use this convention in your macros, to alert a user that the console stack has been cleared.

Top of File and End of File

When an edit macro is invoked and the current line pointer is positioned at the top of the file or at the end of the file, the editor stacks a token in the console stack. If the line pointer is at the top of the file, the token stacked is "TOF"; if the line pointer is at the end of the file the token stacked is "EOF." If you write an edit macro that does not check the status of the console stack, and the macro is invoked from the top or the end of the file, you receive the message:

```
?EDIT: TOF
  or
?EDIT: EOF
```

The editor does not recognize these tokens as valid subcommands.

You may want to use these tokens to test whether the EXEC is invoked from the top or end of the file. If you want to clear these tokens in case the macro has been invoked from the top or end of the file, you might use the statement:

```
&IF &READFLAG EQ STACK &READ VARS
```

which clears the token from the stack.

Stacking LIFO

If you do not want to clear the console stack when you execute an edit macro, you can stack all of the subcommands using the LIFO (last-in first-out) operand of the &STACK and &BEGSTACK control statements. For example, suppose $FORMAT is the name of the following edit macro:

```
&BEGSTACK LIFO
TABSET 3 10 71
TRUNC 71
PRESERVE
&END
```

When this edit macro is executed, the subcommands are placed in the console stack in front of any existing lines. For example, if this macro were invoked:

```
$format#input
```

the subcommands would execute in the following order: PRESERVE, TRUNC, TABSET, INPUT. If the subcommands were stacked FIFO (first-in first-out), the default, the INPUT subcommand would be the first to execute (since it is the first command in the stack) and the remaining subcommands would be read into the file as input lines.

Error Situations

If a CMS EXEC processing error occurs during the execution of an edit macro, the editor clears the console stack and issues the "STACKED LINES CLEARED" message. A CMS EXEC processing error is one that causes the error message DMSEXT072E:

```
ERROR IN EXEC FILE filename, LINE nnnn - description
```

These errors cause the CMS EXEC interpreter to terminate processing. Any stacked subcommands are cleared before the editor regains control, so that none of the subcommands are executed, and the file remains unchanged.

You should also ensure that any error handling routines in your edit macros clear the stack if an error occurs. Otherwise, the editor may begin reading invalid data lines from the stack and attempt to execute them as EDIT subcommands.

You should not interrupt the execution of an edit macro by using the Attention or Enter key, and then entering a command or data line. Results are unpredictable, and you may inadvertently place unwanted lines in the stack.

If your edit macro contains a CMS command that is invalid in the CMS subset environment, you receive a return code of -2.

The maximum number of lines that you can stack in an edit macro varies according to the amount of free storage that is available to CMS at the time of the stacking request. If you stack too many lines, the editor terminates abnormally.

Notes on Using EDIT Subcommands

You can use any EDIT subcommand in a macro file, and there is one special subcommand whose use only has meaning in a macro: the STACK subcommand. For the most part, there is not any difference between executing an EDIT subcommand from the edit environment, or from an EXEC edit macro. You do have to remember, however, that if you want a variable symbol on a subcommand line, you must stack that subcommand using the &STACK control statement rather than following an &BEGSTACK control statement.

Listed below are some notes on using various EDIT subcommands in your macro files. You may find these notes useful when you design your own macros.

PRESERVE, VERIFY, and RESTORE: Often, you may want to create an edit macro that alters the characteristics of a file (format, tab settings, and so on). To ensure that the original characteristics are retained when the macro has finished executing, you can stack the PRESERVE subcommand as the first subcommand in the stack, and the RESTORE subcommand as the last subcommand in the stack:

```
&BEGSTACK
PRESERVE
CASE M
I A lowercase line
RESTORE
&END
```

The PRESERVE and RESTORE subcommands save and reinitialize the settings for the CASE, FMODE, FNAME, IMAGE, LINEMODE, LONG, RECFM, SERIAL, SHORT, TABSET, TRUNC, VERIFY, and ZONE subcommands.

In an edit macro that issues many subcommands that display lines in response to CHANGE or LOCATE subcommands, you may want to turn the verification setting to OFF to suppress displays during the execution of the edit macro:

```
&BEGSTACK
PRESERVE
VERIFY OFF
.
.
.
RESTORE
&END
```

You would particularly want to turn verification off for a macro that executes in a loop or that issues a global request. If you want a line or series of lines displayed, you can use the TYPE subcommand.

If you have verification set off in an edit macro, then when you execute it you may not receive any indication that the edit macro completed execution. The keyboard unlocks to accept your next EDIT subcommand from the terminal. To indicate that the macro is finished, you can stack, as the last subcommand in the procedure, a TYPE subcommand, to display the current line. Or, if you write an edit macro that terminates when an end-of-file condition occurs the EOF: message issued by the editor may indicate the completion of the macro.

INPUT, REPLACE: To change from edit mode to input mode in an edit macro, you can use the INPUT and REPLACE subcommands. In a fixed-length EXEC file, you must stack these subcommands using the &STACK control statement:

```
&STACK INPUT
  -- or --
&STACK REPLACE
```

If you use either of these subcommands following an &BEGSTACK control statement, the subcommand line is padded with blanks to the line length and the result is a line of blanks inserted into the file.

In a variable-length EXEC file, lines are not padded with blanks, so the INPUT and REPLACE subcommands with no data line execute the same following an &BEGSTACK control statement as they do when stacked with the &STACK control statement.

Going From Input Mode to Edit Mode: To stack a null line in an edit macro, to cause the editor to leave input mode, you must use the &STACK control statement with no other tokens, as follows:

```
&STACK
```

CHANGE, DSTRING, LOCATE: If you want to use the CHANGE, DSTRING, or LOCATE subcommands in an EXEC, you must take into account that when you stack any of these subcommands using the &STACK control statement, all of the character strings on the line are truncated or padded to eight characters. Also, if you want to use a variable value for a character string, you are limited to eight characters, all uppercase.

For example, if a macro is used to locate a character string and delete the line on which it appears, the LOCATE subcommand has a variable symbol:

```
&STACK LOCATE /&1
&STACK DEL
```

IMAGE, TABSET, OVERLAY: The TABSET and OVERLAY subcommands allow you to set margins and column stops for records in a file and to overlay character strings in particular positions. For example, the following macro places a vertical bar in columns 1, 15, 40, and 60 for all records in the file from the current line to the end of the file:

```
&BEGSTACK
PRESERVE
IMAGE ON
TABSET 1 15 40 60
REPEAT *
0 |->|->|->|
RESTORE
&END
```

In the above example, the "->" symbol represents a tab character (X'05'). To create this EXEC, you can either issue the EDIT subcommand:

```
image off
```

and use the Tab key (or equivalent) on your terminal when you enter the line, or you can enter some other character and use the ALTER subcommand to alter that character to a X'05'.

If you want to overlay only one character string in a particular position in a file, you can use the TABSET subcommand to set that column position as the left margin, and then use the OVERLAY subcommand, as follows:

```
&CONTROL OFF
&BEGSTACK
PRESERVE
VERIFY OFF
TRUNC *
TABS 72
&END
&STACK REPEAT &1
&BEGSTACK
OVERLAY C
RESTORE
&END
```

If you name this file $CONT EXEC, and if you invoke it with the line:

```
$cont 3
```

then the OVERLAY subcommand is executed on three successive lines, to place the continuation character "C" in column 72.

The STACK Subcommand

The STACK subcommand allows you to stack up to 25 lines from a file in the console stack. The lines are not deleted from the file, but the line pointer is moved to point to the last line stacked.

You can also use the STACK subcommand to stack EDIT subcommands. You might do this if there were subcommands that you wanted to place in the stack to execute after all the subcommands stacked by the EXEC had executed.

These techniques are used in the two edit macros that are distributed with the VM/SP system: $MOVE and $DUP. If you want to examine these files for examples of how to use the STACK subcommand, you can display the files by entering, from the CMS environment:

```
type $move exec *
type $dup exec *
```

An additional use of the STACK subcommand is shown in "An Annotated Edit Macro."

An Annotated Edit Macro

The edit macro shown below, $DOUBLE, can be used to double space a CMS file. Regardless of where the current line pointer is, a blank line is inserted in the file following every existing line. The statements in the edit macro are separated into groups; the number to the left of a statement or group of statements indicates an explanatory note. The numbers are not part of the EXEC file.

```
1     &CONTROL OFF

2     &IF &INDEX = 1 &IF &1 = ? &GOTO -TELL

3     &IF &INDEX = 1 &IF &1 = TWO &GOTO -LOOP

4     &IF &INDEX NE 0 &GOTO -TELL

5     &IF &READFLAG EQ STACK &READ VARS &GARB

6     &STACK
      &STACK PRESERVE
      &STACK VERIFY OFF

7     &STACK BOTTOM
      &STACK I XXXXXXXX
      &STACK TOP

8     -LOOP
      &BEGSTACK
      NEXT
      STACK 1
      INPUT
      &END

9     &READ ARGS
      &IF .&1 = . &SKIP
      &IF &1 EQ XXXXXXXX &SKIP 2

10    -ENDLOOP &STACK $DOUBLE TWO

11    &EXIT

12    DESBUF
      &BEGSTACK
      UP 2
      DEL 3
      TYPE
      RESTORE
      &END

      &EXIT

13    -TELL
      &IF &READFLAG EQ STACK &READ VARS
      &BEGTYPE
      CORRECT FORM IS: $DOUBLE

      THIS EXEC DOUBLE SPACES A FILE BY INSERTING
      A BLANK LINE FOLLOWING EVERY LINE IN THE FILE
      EXCEPT THE LAST.
      &END
```

Explanation by number

1. The &CONTROL statement suppresses the display of CMS commands, in this case, the DESBUF command.

2. The first &IF checks that there is only one operand passed in the $DOUBLE command. The second &IF checks whether $DOUBLE has been invoked with a question mark (?). If both &IFs are true, control is passed to the statement at the label -TELL. &TYPE control statements at -TELL explains what the macro does.

3. The second &IF statement checks whether $DOUBLE has been invoked with the argument TWO, which indicates that the macro has executed itself, so the subcommands that initialize the file are stacked only once.

4. There are three ways to properly invoke this edit macro: with a ?, with the argument TWO, or with no arguments. The third &IF statement checks for the (no arguments) condition; if the macro is invoked any other way, control is passed to the label -TELL, which explains the macro usage.

5. The &READFLAG special variable is checked. If $DOUBLE is executed at the top or at the end of the file, the token TOF or EOF is in the stack, and should be read out.

6. A null line is placed in the console stack for loop control (see Note 9.) The PRESERVE and VERIFY subcommands are stacked so that the editor does not display each line in the file as it executes the stacked subcommands.

7. The BOTTOM, INPUT, and TOP subcommands initialize the file by placing a marker at the bottom of the file, and then positioning the current line pointer at the top of the file.

8. The NEXT, STACK, and INPUT subcommands are going to be repeated for each line in the file. The INPUT subcommand with no data line stacks a null line. Note that in order for $DOUBLE to execute this subcommand properly, $DOUBLE EXEC must have fixed-length records. Each line is stacked, with the STACK subcommand; this stacked line is checked in the read loop (Note 9). When the stacked line is equal to the marker, XXXXXXXX, it indicates that the end of the file has been reached.

9. These lines check for an end of file, which occurs when the line containing the marker is read. The first time this loop is executed, the stack contains the null line (statement 6), so the check for the marker is skipped.

10. The last subcommand stacked is $DOUBLE TWO, which re-invokes $DOUBLE, but causes it to skip the first sequence of subcommands.

11. The &EXIT statement causes an exit from $DOUBLE, so that all the EDIT subcommand stacked thus far are executed.

12. When the marker is read in, the EXEC clears the stack, moves the current line pointer to point to the null line added above the marker, and deletes that line, the marker, and the null line that was inserted following the marker. The RESTORE subcommand restores editor settings.

13. This edit macro is self-documenting. If $DOUBLE is invoked with a question mark, or invoked with an argument, information regarding its proper use is displayed.

User-Written Edit Macros

You can create the edit macros shown below, for your own use in CMS. You may want to refer to them as examples when you are learning to write your own macros. The macros have not been formally tested by IBM; they are presented for your convenience only.

$MACROS

The $MACROS edit macro verifies the existence of and describes the usage of edit macros. If you enter:

```
$macros
```

it lists the filenames of all the edit macros on your accessed disks. If you enter:

```
$macros name1 name2
```

it displays, for each valid macro name entered, the macro format and usage. (This macro assumes that all macros have been designed to respond to a ? request.) The format of the $MACROS edit macro instruction is:

$MACROS	`[filename1 [filename2 [filenamen]]]`

`filename`
is the filename(s) of macro files whose usage is to be displayed. If filename is omitted, the filenames of all available macro files are listed.

To create $MACROS, enter:

```
edit $macros exec
```

and in input mode, enter the following:

```
&CONTROL OFF
&IF &INDEX EQ 1 &IF &1 EQ ? &GOTO -TELL
&IF &INDEX GT 0 &GOTO -PARTIC
*
&BEGTYPE ALL
EXEC FILES STARTING WITH A DOLLAR-SIGN ARE AS FOLLOWS.
FOR INFORMATION ON ONE OR MORE OF THEM, TYPE:
$MACROS FILENAME1 <FILENAME2>
&END
LISTF $* EXEC * (NOHEADER FNAME)
&EXIT
*
-PARTIC &TRIP = 0
&INDEX1 = 0
*
&LOOP -ENDLOOP &INDEX
&INDEX1 = &INDEX1 + 1
&SUB = &SUBSTR &&INDEX1 1 1
&IF &SUB EQ $ &GOTO -STATIT
     &TYPE &&INDEX1 IS INVALID
&TRIP = 1
&GOTO -ENDLOOP
-STATIT STATE &&INDEX1 EXEC *
&IF &RETCODE EQ 0 &GOTO -CALLIT
&TYPE &&INDEX1 NOT FOUND
&TRIP = 1
&GOTO -ENDLOOP
-CALLIT EXEC &&INDEX1 ?
-ENDLOOP
*
&EXIT &TRIP
*
-TELL &BEGTYPE
'$MACROS' HANDLES THE '$MACROS' REQUEST.
TYPE '$MACROS' ALONE FOR MORE INFORMATION.
&END
&EXIT
```

$MARK

The $MARK edit macro inserts from one to six characters, starting with the current line and in the column specified, for a specified number of records. If there is data already in the columns specified, it is overlayed. If you enter:

```
$mark
```

the macro places an asterisk (*) in column 72 of the current line. If you enter:

```
$mark 10 30 abc
```

the macro places the string ABC beginning in column 30 in each of ten records, beginning with the current record. The format of the $MARK edit macro instruction is:

where:

n

indicates the number of consecutive lines, starting with the record currently being pointed to, that will be marked. If n is not specified, 1 is assumed, and the other default values are also assumed.

col
: indicates the starting column in each record where the character string is to be inserted. The default is column 72.

char
: indicates from one to six characters to be inserted in each record. The default is an asterisk (*).

To create $MARK, enter:

```
edit $mark exec
```

and in input mode, enter the following:

```
&CONTROL OFF
&IF &INDEX EQ 1 &IF &1 EQ ? &GOTO -TELL
&IF &INDEX GT 3 &GOTO -BADPARM
&INDEX1 = 1
&IF &INDEX GT 0 &INDEX1 = &1
&IF &INDEX1 LT 0 &GOTO -BADPARM
&INDEX2 = 72
&IF &INDEX GT 1 &INDEX2 = &2
&IF &INDEX2 LT 0 &GOTO -BADPARM
&IF &INDEX2 GT 133 &GOTO -BADPARM
&CHAR = *
&IF &INDEX EQ 3 &CHAR = &3
&LEN3 = &LENGTH &CHAR
&IF &LEN3 GT 6 &GOTO -BADPARM
&STACK LIFO RESTORE
&STACK LIFO OVERLAY &CHAR
&STACK LIFO REPEAT &INDEX1
&STACK LIFO TABS &INDEX2
&BEGSTACK LIFO
IMAGE ON
TRUNC *
VERIFY OFF
LONG
PRESERVE
&END
&EXIT
*
-BADPARM &BEGTYPE
INVALID $MARK OPERANDS
&END
&EXIT 1
*
-TELL &BEGTYPE
CORRECT FORM IS:  $MARK  <N  <COL  <CHAR>>>
PUTS A 1-6 CHARACTER  STRING IN COLUMN 'COL' OF 'N' LINES, STARTING
WITH THE CURRENT LINE.  THE NEW CURRENT LINE IS THE LAST LINE
MARKED.  DEFAULTS ARE:  N=1;  COL=72;  CHAR=*.
&END
&EXIT
```

$POINT

The $POINT edit macro positions the current line pointer at the specified line number. The line numbers must be in columns 73 through 80 and padded with zeros. For example, if you enter:

```
$point 800
```

the current line pointer is positioned at the line that has the serial number 00000800 in columns 73 through 80. The format of the $POINT macro instruction is:

$POINT	key

where:

key
: is a one- to eight-character line number. If the specified key is less than eight characters long, it is padded with leading zeros.

To create $POINT, enter:

```
edit $point exec
```

and in input mode, enter the following:

```
&CONTROL OFF
&IF &INDEX EQ 0 &GOTO -TELL
&IF &INDEX EQ 1 &IF &1 EQ ? &GOTO -TELL
&IF &INDEX GT 1 &GOTO -BADPARM
&KEYL = &LENGTH &1
&INDEX1 = 8 - &KEYL
&Z = &SUBSTR 00000000 1 &INDEX1
&1 = &CONCAT &Z &1
&STACK LIFO RESTORE
&STACK LIFO FIND &1
&BEGSTACK LIFO
TOP
TABS 73
IMAGE ON
LONG
PRESERVE
&END
&EXIT
*
-BADPARM &BEGTYPE ALL
INVALID $POINT OPERANDS
&END
&EXIT 1
*
-TELL &BEGTYPE ALL
CORRECT FORM IS:  $POINT  KEY
IF 'KEY' CONTAINS LESS THAN 8 CHARACTERS, IT IS PADDED WITH LEADING
ZEROS. THE FILE IS THEN SEARCHED FROM THE TOP FOR 'KEY' IN COLUMNS
73-80.
&END
&EXIT
```

$COL

The $COL edit macro inserts, after the current record in the file, a line containing column numbers (that is, 1, 6, 11, ..., 76). The format of the $COL macro instruction is:

$COL	

No operands are used with $COL. If any arguments are entered, the macro usage is explained.

To create $COL, enter:

```
edit $col exec
```

and in input mode, enter the following:

```
&CONTROL OFF
&IF &INDEX NE 0 &GOTO -TELL
&STACK LIFO RESTORE
&STACK LIFO
&BEGSTACK LIFO ALL
1   6   11   16   21   26   31   36   41   46   51   56   61   66   71   76
&END
&STACK LIFO INPUT
&BEGSTACK LIFO
TRUNC *
VERIFY OFF
LONG
PRESERVE
&END
&EXIT
*
-TELL &BEGTYPE
CORRECT FORM IS:  $COL
INSERTS A LINE INTO THE FILE SHOWING COLUMN NUMBERS.
&END
&EXIT
```

Appendix C. Considerations for Line Mode Terminals

Logical Line Editing Symbols

To aid you in entering command or data lines from your terminal, VM/SP provides a set of logical line editing symbols, which you can use to correct mistakes as you enter lines. Each symbol has been assigned a default character value. These normally are:

Symbol	Character
Logical character delete	@
Logical line end	#
Logical line delete	¢
Logical escape	"

Logical Character Delete

The logical character delete symbol (@) allows you to delete one or more of the previous characters entered. The @ deletes one character per @ entered, including the ¢ and # logical editing characters. For example:

```
ABC#@@ results in AB
ABC@D results in ABD
¢@DEF results in DEF
ABC@@@ deletes the entire string
```

Logical Line End

The logical line end symbol (#) allows you to key in more than one command on the same line, and thus minimizes the amount of time you have to wait between entering commands. You type the # at the end of each logical command line, and follow it with the next logical command line. VM/SP stacks the commands and executes them in sequence. For example, the entry:

```
query blip#query rdymsg#query search
```

is executed in the same way as the entries:

```
query blip
query rdymsg
query search
```

The logical line end symbol also has special significance for the #CP function. Beginning any physical line with #CP indicates that you are entering a command that is to be processed by CP immediately. If you have set a character other than # as your logical line end symbol, you should use that character instead of a #.

Logical Line Delete

The logical line delete symbol (¢) terminals) deletes the entire previous physical line, or the last logical line back to (and including) the previous logical line end (#). You can use it to cancel a line containing many or serious errors. If a # immediately precedes the ¢ sign, only the # sign is deleted, since the # indicates the beginning of a new line, and the ¢ cancels the current line. For example:

- Logical Line Delete:

```
ABC#DEF¢ deletes the #DEF and results in ABC
ABC#¢ results in ABC
ABC#DEF¢#GHI results in ABC#GHI
ABC#DEF¢GHI results in ABCGHI
```

- Physical Line Delete:

```
ABC¢ deletes the whole line
```

Note: When you cancel a line by using the ¢ logical line delete symbol, you do not need to press a carriage return; you can continue entering data on the same line.

Logical Escape

The logical escape symbol (") causes VM/SP to consider the next character entered to be a data character, even if it is normally one of the logical line editing symbols (@, ¢, ", or #). For example:

```
ABC"¢D results in ABC¢D
""ABC"" results in "ABC"
```

If you enter a single logical escape symbol (") as the last character on a line, or on a line by itself, it is ignored.

When you enter logical escape characters in conjunction with other logical editing characters, the results may be difficult to predict. For example, the lines:

```
ABC""@DEF
ABC""@@DEF
```

both result in the line ABCDEF.

Defining Logical Line Editing Symbols

The logical line editing symbols are defined for each virtual machine during VM/SP system generation. If your terminal's keyboard lacks any of these special characters, your installation can define other special characters for logical line editing. You can find out what logical line editing symbols are in effect for your virtual machine by entering the command:

```
cp query terminal
```

The response might be something like:

```
LINEND # , LINEDEL ¢ , CHARDEL @ , ESCAPE "
LINESIZE 130, MASK OFF, APL OFF, ATTN OFF, MODE VM
```

You can use the CP TERMINAL command to change the logical line editing characters for your virtual machine. For example, if you enter:

```
cp terminal linend /
```

Then, the line:

```
input # line / input / #
```

would be interpreted:

```
input # line
input
#
```

The terminal characteristics listed in the response to the CP QUERY TERMINAL command are all controlled by operands of the CP TERMINAL command.

Appendix D. Summary of CMS Commands

Figure D-1 on page D-3 and Figure D-2 on page D-7 contain alphabetical lists of the CMS commands and the functions performed by each. Figure D-1 on page D-3 lists those commands that are available for general use; Figure D-2 on page D-7 lists the commands used by system programmers and system support personnel who are responsible for generating, maintaining, and updating VM/SP. Unless otherwise noted, CMS commands are described in *VM/SP CMS Command and Macro Reference*

Code	Meaning
VSE PP	Indicates that this command invokes a VSE Program Product, available from IBM for a license fee.
EREP	Indicates that this command is described in *VM/SP OLTSEP and ERROR Recording Guide*. Further details on the operands used by this command are contained in *OS/VS Environmental Recording Editing and Printing (EREP) Program.*
IOCP UG	indicates that this command is described in the *Input/Output Configuration Program User's Guide and Reference.*
IPCS	Indicates that this command is a part of the Interactive Problem Control System (IPCS) and is described in *VM/370 IPCS User's Guide.*
Op Gd	Indicates that this command is described in the *VM/SP Operator's Guide.*
OS PP	Indicates that this command invokes an OS program product, available from IBM for a license fee.
SPG	Indicates that this command is described in the *VM/SP System Programmer's Guide.*
PLAN	Indicates that this command is described in the *VM/SP Planning Guide and Reference.*
INSTALL	Indicates that this command is described in the *VM/SP Installation Guide.*

There are ten commands called Immediate commands that are handled in a different manner from the other commands listed in Figure D-1 and Figure D-2. They may be entered while another command is executing by pressing the Attention key (or its equivalent) and are executed immediately.

The Immediate commands are:

HB	Halt batch execution
HI	Halt Interpretation
HO	Halt tracing
HT	Halt typing
HX	Halt execution
RO	Resume tracing
RT	Resume typing
SO	Suspend tracing
TE	Trace end
TS	Trace start

You can define your own immediate commands by using any of the following:

- the IMMCMD macro in an assembler language program
- the IMMCMD command within an EXEC (CMS EXEC, EXEC 2, System Product interpreter).
- NUCXLOAD command with the IMMCMD option specified.

Command	Code	Usage
ACCESS		Identify direct access space to a CMS virtual machine, create extensions and relate the disk space to a logical directory.
AMSERV		Invoke access method services utility functions to create, alter, list, copy, delete, import, or export VSAM catalogs and data sets.
ASSEMBLE		Assemble assembler language source code.
ASSGN		Assign or unassign a CMS/DOS system or programmer logical unit for a virtual I/O device.
CATCHECK		Allows a CMS VSAM user (with or without DOS set ON) to invoke the VSE/VSAM Catalog Check Service Aid to verify a complete catalog structure.
CMDCALL		Converts EXEC 2 extended plist function calls to CMS extended plist command calls.
CMSBATCH		Invoke the CMS batch facility.
COMPARE		Compare records in CMS disk files.
CONWAIT		Causes a program to wait until all pending terminal I/O is complete.
COPYFILE		Copy CMS disk files according to specifications.
CP		Enter CP commands from the CMS environment.
DDR		Perform backup, restore, and copy operations for disks.
DEBUG		Enter DEBUG subcommand environment.
DEFAULTS		Set or display default options for the commands: FILELIST, NOTE, RDRLIST, RECEIVE, PEEK, SENDFILE, and TELL.
DESBUF		Clears the program stack and the terminal input buffers.
DISK		Perform disk-to-card and card-to-disk operations for CMS files.
DLBL		Define a VSE filename or VSAM ddname and relate that name to a disk file.
DOSLIB		Delete, compact, or list information about the phases of a CMS/DOS phase library.
DOSLKED		Link-edit CMS text decks or object modules from a VSE relocatable library and place them in executable form in a CMS/DOS phase library.
DOSPLI	VSE PP	Compile DOS PL/I source code under CMS/DOS.
DROPBUF		Eliminate a program stack buffer.
DSERV		Display information contained in the VSE core image, relocatable, source, procedure, and transient directories.
EDIT		Invoke the VM/SP System Product editor in CMS editor (EDIT) compatibility mode to create or modify a disk file.
ERASE		Delete CMS disk files.
ESERV		Display, punch or print an edited (compressed) macro from a VSE source statement library (E sublibrary).
EXEC		Execute special procedures made up of frequently used sequences of commands.

Figure D-1 (Part 1 of 4). CMS Command Summary

Command	Code	Usage
EXECIO		Do I/O operations between a device and the program stack.
EXECOS		Reset the OS and VSAM environments under CMS without returning to the interactive environment.
EXECUPDT		Produces an updated executable version of a System Product interpreter source program.
FCOBOL	VSE PP	Compile DOS/VS COBOL source code under CMS/DOS.
FETCH		Fetch a CMS/DOS or VSE executable phase.
FILEDEF		Define an OS ddname and relate that ddname to any device supported by CMS.
FILELIST		List information about CMS disk files, with the ability to edit and issue commands from the list.
FINIS		Close an open file.
FORMAT		Prepare disks in CMS fixed block format.
GENDIRT		Fill in auxiliary module directories.
GENMOD		Generate nonrelocatable CMS files (MODULE files).
GLOBAL		Identify specific CMS libraries to be searched for macros, copy files, missing subroutines, LOADLIB modules, or DOS executable phases.
GLOBALV		Set, maintain, and retrieve a collection of named variables.
HELP		Display information about CP, CMS, or user commands, EDIT, XEDIT, or DEBUG subcommands, System Product interpreter, EXEC, and EXEC2 control statements, and descriptions of CMS and CP messages.
IDENTIFY		Display or stack userid, nodeid, rscsid, date, time, time zone, and day of the week.
IMMCMD		Use the IMMCMD command to establish or cancel immediate commands from within an EXEC.
INCLUDE		Bring additional TEXT files into storage and establish linkages.
IOCP	IOCP UG	Invoke the Input/Output Configuration Program
LABELDEF		Specify standard HDR1 and EOF1 tape label description information for CMS, CMS/DOS, and OS simulation.
LISTDS		List information about data sets and space allocation on OS, DOS, and VSAM disks.
LISTFILE		List information about CMS disk files.
LISTIO		Display information concerning CMS/DOS system and programmer logical units.
LKED		Link edit a CMS TEXT file or OS object module into a CMS LOADLIB.
LOAD		Bring TEXT files into storage for execution.
LOADLIB		Maintain CMS LOADLIB libraries.
LOADMOD		Bring a single MODULE file into storage.
MACLIB		Create or modify CMS macro libraries.
MAKEBUF		Create a new program stack buffer.

Figure D-1 (Part 2 of 4). CMS Command Summary

Command	Code	Usage
MODMAP		Display the load map of a MODULE file.
MOVEFILE		Move data from one device to another device of the same or a different type.
NAMEFIND		Display/stack information from a NAMES file. (default 'userid NAMES').
NAMES		Display a menu to create, display or modify entries in a 'userid NAMES' file. (The menu is available only on display terminals.)
NOTE		Prepare a 'note' for one or more computer users, to be sent via the SENDFILE command.
NUCXDROP		Delete specified nucleus extensions.
NUCXLOAD		Load a nucleus extension.
NUCXMAP		Identify existing nucleus extensions.
OPTION		Change the DOS/VS COBOL compiler (FCOBOL) options that are in effect for the current terminal session.
OSRUN		Load, relocate, and execute a load module from a CMS LOADLIB or OS module library.
PEEK		Display a file that is in your virtual reader without reading it onto disk.
PRINT		Spool a specified CMS file to the virtual printer.
PSERV		Copy a procedure from the VSE procedure library onto a CMS disk, display the procedure at the terminal, or spool the procedure to the virtual punch or printer.
PUNCH		Spool a copy of a CMS file to the virtual punch.
QUERY		Request information about a CMS virtual machine.
RDR		Generate a return code and either display or stack a message that identifies the characteristics of the next file in your virtual reader.
RDRLIST		Display information about files in your virtual reader with the ability to issue commands from the list.
READCARD		Read data from spooled card input device.
RECEIVE		Read onto disk a file or note that is in your virtual reader.
RELEASE		Make a disk and its directory inaccessible to a CMS virtual machine.
RENAME		Change the name of a CMS file or files.
RESERVE		Use the RESERVE command to allocate all available blocks of a 512-, 1K-, 2K-, or 4K-byte block formatted mini-disk to a unique CMS file.
RSERV		Copy a VSE relocatable module onto a CMS disk, display it at the terminal, or spool a copy to the virtual punch or printer.
RUN		Initiate series of functions to be performed on a source, MODULE, TEXT, or EXEC file.
SENDFILE		Send files or notes to one or more computer users, attached locally or remotely, by issuing the command or by using a menu.(display terminal only)
SENTRIES		Determine the number of lines currently in the program stack.

Figure D-1 (Part 3 of 4). CMS Command Summary

Command	Code	Usage
SET		Establish, set, or reset CMS virtual machine characteristics.
SETPRT		Load a virtual 3800 printer.
SORT		Arrange a specified file in ascending order according to sort fields in the data records.
SSERV		Copy a VSE source statement book onto a CMS disk, display it at the terminal, or spool a copy to the virtual punch or printer.
START		Begin execution of programs previously loaded (OS and CMS) or fetched (CMS/DOS).
STATE		Verify the existence of a CMS disk file.
STATEW		Verify a file on a read/write CMS disk.
SVCTRACE		Record information about supervisor calls.
SYNONYM		Invoke a table containing synonyms you have created for CMS and user-written commands.
TAPE		Perform tape-to-disk and disk-to-tape operations for CMS files, position tapes, and display or write VOL1 labels.
TAPEMAC		Create CMS MACLIB libraries directly from an IEHMOVE-created partitioned data set on tape.
TAPPDS		Load OS partitioned data set (PDS) files or card image files from tape to disk.
TELL		Send a message to one or more computer users who are logged on to your computer or to one attached to yours via RSCS.
TXTLIB		Generate and modify text libraries.
TYPE		Display all or part of a CMS file at the terminal.
UPDATE		Make changes in a program source file as defined by control cards in a control file.
VSAPL	OS PP	Invoke VS APL interface in CMS.
XEDIT		Invoke the VM/SP System Product editor to create or modify a disk file.

Figure D-1 (Part 4 of 4). CMS Command Summary

Command	Code	Usage
ASM3705	INSTALL	Assemble 370x source code.
ASMGEND	INSTALL	Regenerate the VM/SP assembler command modules.
CMSGEND	INSTALL	Generate a new CMS disk-resident module from updated TEXT files.
CPEREP	EREP	Format and edit system error records for output.
DIRECT	PLAN	Set up VM/SP directory entries.
DOSGEN	INSTALL	Load and save CMSDOS and INSTVSAM shared segments.
DUMPSCAN	IPCS	Provide interactive analysis of CP abend dumps.
GEN3705	INSTALL	Generate an EXEC file that assembles and link-edits the 370x control program.
GENERATE	INSTALL	Update VM/SP or the VM/SP directory, or generate a new standalone copy of a service program.
NCPDUMP	OP Gd, SPG	Process CP spool reader files created by 370x dumping operations.
PRB	IPCS	Update IPCS problem status.
PROB	IPCS	Enter a problem report in IPCS.
PROP	OP Gd	Provide Programmable Operator capability.
SAMGEN	INSTALL	Load and save the CMSBAM shared segment.
SAVENCP	INSTALL, SPG	Read 370x control program load into virtual storage and save an image on a CP-owned disk.
SETKEY	SPG	Assign storage protect keys to storage assigned to named systems.
STAT	IPCS	Display the status of reported system problems.
TRAPRED	OP Gd	Allow the data collected by CPTRAP to be displayed or printed.
VMFDOS	INSTALL	Create CMS files for VSE modules from VSE library distribution tape or SYSIN tape.
VMFDUMP	OP Gd, IPCS	Format and print system abend dumps; under IPCS, create a problem report.
VMFLOAD	INSTALL	Generate a new CP, CMS, or RSCS module.
VSAMGEN	INSTALL	Load and save CMSVSAM and CMSAMS shared segments.
VSEVSAM	INSTALL	Build a VSE/VSAM maclib containing the supported VSE/VSAM macros as well as the following VSE macros: CDLOAD, CLOSE, CLOSER,GET, OPEN, OPENR, and PUT.
ZAP	OP Gd, SPG	Modify or dump LOADLIB, TXTLIB, or MODULE files.

Figure D-2. Summary of CMS Commands for System Programmers

Appendix E. Summary of CP Commands

Figure E-1 describes the CP command privilege classes. Figure E-2 on page E-2 summarizes the CP commands and gives a brief description of each.

Class	User and Function
A[1]	***Primary System Operator***: The class A user controls the VM/SP system. Class A is assigned to the user at the VM/SP system console during IPL. The primary system operator is responsible for the availability of the VM/SP system and its communication lines and resources. In addition, the class A user controls system accounting, broadcast messages, virtual machine performance options and other command operands that affect the overall performance of VM/SP. **Note:** The class A system operator who is automatically logged on during CP initialization is designated as the primary system operator.
B[1]	***System Resource Operator***: The class B user controls all the real resources of the VM/SP system, except those controlled by the primary system operator and spooling operator.
C[1]	***System Programmer***: The class C user updates certain functions of the VM/SP system.
D[1]	***Spooling Operator***: The class D user controls spool data files and specific functions of the system's unit record equipment.
E[1]	***System Analyst***: The class E user examines and saves certain data in the VM/SP storage area.
F[1]	***Service Representative***: The class F user obtains, and examines, in detail, certain data about input and output devices connected to the VM/SP system.
G[2]	***General User***: The class G user controls functions associated with the execution of his virtual machine.
Any[2]	The Any classification is given to certain CP commands that are available to any user. These are primarily for the purpose of gaining and relinquishing access to the VM/SP system.
H	Reserved for IBM use.

Figure E-1. CP Privilege Class Descriptions

1 Described in the *VM/SP Operator's Guide*.

2 Described in the *VM/SP CP Command Reference for General Users*.

Figure E-2 contains an alphabetical list of the CP commands, the privilege classes which may execute the command, and a brief statement about the use of each command.

Command	Privilege Class	Usage
*	any	Annotate the console sheet.
#CP	any	Execute a CP command while remaining in the virtual machine environment.
ACNT	A	Create accounting records for logged on users and reset accounting data, and close the spool file that is accumulating accounting records.
ADSTOP	G	Halt execution at a specific virtual machine instruction address.
ATTACH	B	Attach a real device to a virtual machine. Attach a DASD for CP control. Dedicate all devices on a particular channel to a virtual machine.
ATTN	G	Make an attention interruption pending for the virtual machine console.
AUTOLOG	A,B	Automatically log on a virtual machine and have it operate in disconnect mode.
BACKSPAC	D	Restart or reposition the output of a unit record spooling device.
BEGIN	G	Continue or resume execution of the virtual machine at either a specific storage location or at the address in the current PSW.
CHANGE	D,G	Alter one or more attributes of a closed spool file CLOSE G Terminate spooling operations on a virtual card reader, punch, printer, or console.
COUPLE	G	Connect channel-to-channel adapters.
CP	any	Execute a CP command while remaining in the CMS virtual machine environment.
CPTRAP	C	Create a file of selected trace table, CP interface, and virtual machine interface entries for problem determination.
DCP	C,E	Display real storage at terminal.
DEFINE	G	Reconfigure your virtual machine.
	B	Redefine the usage of SYSVIRT and VIRTUAL 3330V devices.
DETACH	B	Disconnect a real device from a virtual machine.
	B	Detach a DASD volume from CP.
	B	Detach a channel from a specific user.
	G	Detach a virtual device from a virtual machine.
	G	Detach a channel from your virtual machine.
DIAL	any	Connect a terminal or display device to the virtual machine's virtual communication line.
DISABLE	A,B	Disable 2701/2702/2703, 370X in EP mode, and 3270 local communication lines.
DISCONN	any	Disconnect your terminal from your virtual machine.
DISPLAY	G	Display virtual storage on your terminal.
DMCP	C,E	Dump the specified real storage location on your virtual printer.

Figure E-2 (Part 1 of 4). CP Command Summary

Command	Privilege Class	Usage
DRAIN	D	Halt operations of specified spool devices upon completion of current operation.
DUMP	G	Print the following on the virtual printer: virtual PSW, general registers, floating-point registers, storage keys, and contents of specified virtual storage locations.
ECHO	G	Test terminal hardware by redisplaying data entered at the terminal.
ENABLE	A,B	Enable communication lines.
EXTERNAL	G	Simulate an external interruption for a virtual machine and return control to that machine.
FLUSH	D	Cancel the current file being printed or punched on a specific real unit record device.
FORCE	A	Cause logoff of a specific user.
FREE	D	Remove spool HOLD status.
HALT	A	Terminate the active channel program on specified real device.
HOLD	D	Defer real spooled output of a particular user.
INDICATE	A,E,G	Indicate resource utilization and contention.
IPL	G	Simulate IPL for a virtual machine.
LINK	G	Provide access to a specific DASD by a virtual machine.
LOADBUF	D	Load real UCS/UCSB or FCB printer buffers.
LOADVFCB	G	Load virtual forms control buffer for a virtual 3203 or 3211 printer.
LOCATE	C,E	Find CP control blocks.
LOCK	A	Bring virtual pages into real storage and lock them; thus, excluding them from future paging.
LOGOFF	any	Disable access to CP.
LOGON	any	Provide access to CP.
MESSAGE	A,B,any	Transmit messages to other users.
MIGRATE	A	Allows the operator to migrate pages either for the entire system or just one user.
MONITOR	A,E	Trace events of the real machine and record system performance data.
MSGNOH	B	Send a specified message, without the standard message header, from one virtual machine to another.
NETWORK	A,B	Load, dump, and control the operation of the 370X control program. Control the operation of 3270 remote devices.
NOTREADY	G	Simulate "not ready" for a device to a virtual machine.
ORDER	D,G	Rearrange closed spool files in a specific order.
PER	A,B,C,D, E,F,G	Monitor certain events in the user's virtual machine as they occur during program execution.
PURGE	D,G	Remove closed spool file from system.

Figure E-2 (Part 2 of 4). CP Command Summary

Command	Privilege Class	Usage
QUERY	A,B,C,D, E,F,G	Request information about machine configuration and system status.
QVM	A	Request the transition from VM/SP to the V=R virtual machine running in native mode.
READY	G	Simulate device end interruption for a virtual device.
REPEAT	D	Repeat (a specified number of times) printing or punching of a specific real spool output file.
REQUEST	G	Make an attention interruption pending for the virtual machine console.
RESET	G	Clear and reset all pending interruptions for a specified virtual device and reset all error conditions.
REWIND	G	Rewind (to load point) a tape and ready a tape unit.
SAVESYS	E	Save virtual machine storage contents, registers and PSW.
SCREEN	G	Control color and extended highlight attributes of the screen.
SET	A,B,E,F,	Operator--establish system parameters.
	G	User--control various functions within the virtual machine.
SHUTDOWN	A	Terminate all VM/SP functions and checkpoint CP system for warm start.
SLEEP	any	Place virtual machine in dormant state.
SMSG	G	Send special message to appropriate virtual machine.
SPACE	D	Force single spacing on printer.
SPMODE	A	Establish or reset the single processor mode environment.
SPOOL	G	Alter spooling control options; direct a file to another virtual machine or to a remote location via the RSCS virtual machine.
SPTAPE	D	Dump output spool files on tape or load output spool files from tape.
START	D	Start spooling device after draining or changing output classes.
STCP	C	Change the contents of real storage.
STORE	G	Alter specified virtual storage locations and registers.
SYSTEM	G	Simulate RESET, CLEAR STORAGE, and RESTART buttons on a real system console.
TAG	G	Specify variable information to be associated with a spool file or output unit record device. Interrogate the current TAG text setting of a given spool file or output unit record device.
TERMINAL	G	Define or redefine the input and attention handling characteristics of your virtual console.
TRACE	G	Trace specified virtual machine activity at your terminal, spooled printer, or both.
TRANSFER	D,G	Transfer input files to or reclaim input files from a specified user's virtual card reader.
UNLOCK	A	Unlock previously locked page frames.
VARY	B	Mark a device unavailable or available.

Figure E-2 (Part 3 of 4). CP Command Summary

Command	Privilege Class	Usage
VMDUMP	G	Dump virtual machine when issued with the VM/IPCS Extension program product.
WARNING	A,B	Transmit a high priority message to a specified user or to all users.

Figure E-2 (Part 4 of 4). CP Command Summary

Appendix F. Sample Terminal Sessions

This appendix provides sample terminal sessions showing you how to use:

- The CMS Editor (using context editing), and the CMS COPYFILE, SORT, RENAME, and ERASE commands
- The CMS Editor (using line-number editing)
- CMS OS simulation to create, assemble, and execute a program using OS macros in the CMS environment
- CMS VSE/AF simulation to create, assemble, and execute a program using macros in the CMS/DOS environment.
- Access method services under CMS, to create VSAM catalogs and data spaces, and to use the define and repro functions of AMSERV

Sample Terminal Session Using the CMS Editor and CMS File System Commands

This terminal session shows you how to create a CMS file and make changes to it using the CMS Editor, and then manipulate it using the CMS file system commands, COPYFILE, ERASE, RENAME, and SORT.

Note: Throughout this terminal session whenever a TYPE subcommand or command is issued that results in a display of the entire file, the complete display is not shown; omitted lines are indicated by vertical ellipses (...). When you enter the TYPE command or subcommand, you should see the entire display.

```
1    edit command data
     NEW FILE:
     EDIT:
2    image
     ON
     tabs 1 12 80
     trunc 72
3    input
     INPUT:
     copyfile    copy cms files
     sort        sort cms files in alphameric order by specific columns
     edit        create a cms file
     edit        modify a cms file
     rename      change the name of a cms file
     punch       punch a copy of a cms file on cards
     print       print a cms file
     erase       erase a cms file
     listfile    list information on a cms file
     state       verify the existence of a cms file
     statew      verify the existence of a cms file on a read/write disk
     readcard    read a cms file from your card reader onto disk
     disk dump   punch a cms file in cms disk dump format into your virtual card punch for
4    TRUNCATED
     DISK DUMP  PUNCH A CMS FILE IN CMS DISK DUMP FORMAT INTO YOUR VIRTUAL CA
     disk load   read a disk dump file onto disk
     compare     compare the contents of cms disk files
     tape dump   dump cms files onto tape
     tape load   read cms files onto disk from tape
5
     EDIT:
```

---

1 Use the EDIT command to invoke the CMS Editor to create a file with a filename of COMMAND and a filetype of DATA. Since the file does not exist, the editor issues the message NEW FILE.

2 Check that the image setting is ON. This is the default for all filetypes except SCRIPT. Then, set the logical tab stops for this file at 1, 12, and 80, and set a truncation limit of 72.

3 Enter the subcommand INPUT to enter input mode and begin entering lines in the file. For these input files, you should press the Tab key (or equivalent) on your terminal following each CMS command name. If there is a physical tab stop on your terminal in column 12, the input data appears aligned.

4 The message, TRUNCATED, indicates that the line you just entered exceeded the truncation limit you set for the file (column 72). The editor displays the line, so you can see how much of the line was accepted. Your virtual machine is still in input mode, so continue entering input lines.

5 To get out of input mode, enter a null line (press the Return or Enter key without entering any data). The editor responds with the message EDIT:.

```
5    top
     TOF:
7    type *
     TOF:
     COPYFILE   COPY CMS FILES
        .
        .
        .
     TAPE LOAD  READ CMS FILES ONTO DISK FROM TAPE
8    EOF:
     locate /disk dump
     DISK DUMP  PUNCH A CMS FILE IN CMS DISK DUMP FORMAT INTO YOUR VIRTUAL CA
9    replace disk dump punch a cms file onto cards
     input
     INPUT:
     type        display the contents of a cms file at your terminal
     rename      alter the name of a cms file
     sort        resequence the records in a cms file
     copyfile    reformat a file, by columns
     comprae     verify that two files are identical
10
     EDIT:
     change /rae/are/
     COMPARE    VERIFY THAT TWO FILES ARE IDENTICAL
11   bo
     TAPE LOAD  READ CMS FILES ONTO DISK FROM TAPE
     input
     INPUT:
12
     EDIT:
13   file
     R;
```

---

6 Use the TOP subcommand to position the current line pointer at the top of the file. The editor responds TOF:.

7 Use the TYPE subcommand to display the entire file. Note that all of your input lines are translated to uppercase characters, and that the tab characters you entered have been expanded, so that the first word following each command name begins in column 12.

8 The message EOF: indicates that the end of the file is reached. You can issue the LOCATE subcommand to locate a line. Since you are at the bottom of the file, the editor begins searching from the top of the file. Notice that you can enter the character string you want to locate in lowercase characters; the editor translates it to uppercase to locate the line. The editor displays the line.

9 Use the REPLACE subcommand to replace this line, in a shortened form so that it is not truncated. Remember to enter a tab character after the command name; when you enter the line, the tab stop does not have to be in column 12. Then, use the INPUT subcommand again to resume entering input. The lines that you enter next are written into the file following the DISK DUMP line.

10 When you make a spelling error or other mistake, you may want to correct it immediately. Enter a null line to return to edit mode, and use the CHANGE subcommand to correct the error. In this example, the string RAE is changed to ARE. The editor displays the line as changed.

11 Use the BOTTOM subcommand to move the current line pointer to point to the last line in the file. Enter input mode with the INPUT subcommand.

12 If you enter input mode and decide that you do not want to enter input lines, all you have to do to return to edit mode is enter a null line.

13 To write the file onto disk, use the FILE subcommand. This writes it onto disk using the name with which you invoked the editor, COMMAND DATA. The CMS ready message indicates that you are in the CMS command environment.

```
14   type command data

     COPYFILE   COPY CMS FILES
     SORT       SORT CMS FILES IN ALPHAMERIC ORDER BY SPECIFIC COLUMNS
        .
        .
        .
     TAPE LOAD  READ CMS FILES ONTO DISK FROM TAPE
     R;
15   edit command data
     EDIT:
16      .
        .
        .
     save
     EDIT:
17   fname comm2
     file
     R;
18   copyfile comm2 data a (lowcase
     R;
19   copyfile command data a comm2 data a (ovly specs
     DMSCPY601R ENTER SPECIFICATION LIST:
     1-12 1
     R;
20   type comm2 data

     COPYFILE   Copy cms files
     SORT       Sort cms files in alphameric order by specific columns
     EDIT       Create a cms file
     EDIT       Modify a cms file
     RENAME     Change the name of a cms file
     PUNCH      Punch a copy of a cms file on cards
     PRINT      Print a cms file
     ERASE      Erase a cms file
     LISTFILE   List information on a cms file
21   ht
     R;
```

---

14 To display the entire file at your terminal, use the CMS TYPE command. Note any errors that you made that you might want to correct.

15 Use the EDIT command to edit the file COMMAND DATA again. This time, since the file exists, the editor does not issue the message, NEW FILE:

16 While you are in edit mode, make any changes that you need to; then issue the SAVE subcommand to save these changes, and replace the existing copy of the file onto disk.

17 Use the FNAME subcommand to change the filename of the file to COMM2 (the filetype remains unchanged). When you issue the FILE subcommand this time, the file is written onto disk with the name COMM2 DATA.

18 You can rewrite the entire file, COMM2 DATA in lowercase characters, using the COPYFILE command with the LOWCASE option.

19 The file COMM2 DATA is now all lowercase characters (you can display the file with the TYPE command if you want to verify it). However, the command names, and the first character of the description should be uppercase characters. You can use the COPYFILE command again, to overlay the original uppercase characters of COMMAND DATA in columns 1 through 12 over the lowercase characters in columns 1 through 12 of COMM2 DATA.

20 Use the TYPE command to verify that the COPYFILE command did, in fact, overlay only the columns that you wanted.

21 The HT Immediate command suppresses the display of the remainder of the file; you can see from the first few lines that the format of the file is correct.

```
2   listfile * data
    COMMAND  DATA     A1
    COMM2    DATA     A1
    R;
3   sort comm2 data a command sort a
    DMSSRT604R ENTER SORT FIELDS:
    1 9
    R;
4   type command sort

    COMPARE    Verify that two files are identical
    COMPARE    Compare the contents of cms disk files
       .
       .
       .
    TYPE       Display the contents of a cms file at your terminal

    R;
5   copyfile comm2 data a function data a ( specs
    DMSCPY601R ENTER SPECIFICATION LIST:
    12-72 1 1-9 70
    R;
6   type function data

    Copy cms files                                                  COPYFILE
    Sort cms files in alphameric order by specific columns          SORT
       .
       .
       .
    Read cms files onto disk from tape                              TAPE LOAD
    R;
7   sort function data a functicn sort a
    DMSSRT604R ENTER SORT FIELDS:
    1 70
    R;
    type function sort
    Alter the name of a cms file                                    RENAME
    Change the name of a cms file                                   RENAME
       .
       .
       .
    Verify the existence of a cms file on a read/write disk         STATEW
    R;
```

---

22 The LISTFILE command lists your two files with the filetype of DATA. (If you previously had files with these filetypes, they are also listed.)

23 To sort the file COMM2 DATA into alphabetic order, by command, issue the SORT command. When you are prompted for the sort fields, enter the columns that contain the command names, 1 through 9.

24 The output file from the SORT command is named COMMAND SORT. You can use the TYPE command to verify that the records are now sorted alphabetically by command.

25 To create another copy of the file, this time with the command names on the right and the functional description on the left, use the COPYFILE command with the SPECS option again. To create a file this way, you must know the columns in your input file (COMM2 DATA) and how you want them arranged in your output file (FUNCTION DATA). Columns 1 through 9 contain the command names; columns 12 through 72 contain the descriptions. The specification list entered after the prompting message indicates that columns 12 through 72 should be copied and placed beginning in column 1, and that columns 1 through 9 should be copied beginning in column 70.

26 Verify the COPYFILE operation with the TYPE command.

27 Sort the file FUNCTION DATA so that the functional descriptions appear in alphabetic order. You may also want to display the output file, FUNCTION SORT.

```
28   listfile
     COMMAND  DATA       A1
     COMM2    DATA       A1
     COMMAND  SORT       A1
     FUNCTION DATA       A1
     FUNCTION SORT       A1
     R;
29   erase command data
     R;
30   rename comm2 data a command data a
     R;
     listfile * * a ( label
     FILENAME FILETYPE  FM  FORMAT  LRECL  RECS BLOCKS   DATE     TIME     LABEL
     FUNCTION SORT      A1  F          80    22      3 10/13/75  7:52:03  ABC191
     COMMAND  DATA      A1  F          80    22      3 10/13/75  7:48:52  ABC191
     COMMAND  SORT      A1  F          80    22      3 10/13/75  7:48:15  ABC191
     FUNCTION DATA      A1  F          80    22      3 10/13/75  7:51:37  ABC191
     R;
31   edit function sort
     EDIT:
32   zone
      1 80
     zone 60
33   change /     // *
     Alter the name of a cms file                                    RENAME
     Change the name of a cms file                                   RENAME
        .
        .
        .
     Verify the existence of a cms file on a read/write disk         STATEW
     EOF:
34   top
     TOF:
     find List
35   NOT FOUND
     EOF:
     case
      U
     case m
     find List
     List information on a cms file                                  LISTFILE
```

---

28 If these are the only files on your A-disk, the LISTFILE command entered with no operands produces a list of the files created so far.

29 The file COMM2 was created for a workfile, in case any errors might have happened. Since you no longer need the original file, COMMAND DATA, you can erase it.

30 Use the RENAME command to rename the workfile COMM2 DATA to have the name COMMAND DATA. The LISTFILE command verifies the change.

31 To begin altering the file FUNCTION SORT, invoke the editor again.

32 The ZONE command requests a display of the current zone settings, which are columns 1 and 80. When you issue the command ZONE 60, it changes the settings to columns 60 and 80, so that you cannot modify data in columns 1 through 59.

33 The CHANGE subcommand requests that the first appearance of five consecutive blanks on each line in the file be compressed. The editor displays the results of this CHANGE request by displaying each line changed (which is each line in the file). The net effect is to shift the command column 5 spaces to the left.

34 Position the current line pointer at the top of the file, and then issue a FIND subcommand to move the line pointer to the line that begins with "List".

35 The editor indicates that the line is not found. Checking the current setting for the CASE subcommand, you can see that it is U, or uppercase, which indicates that the editor is translating your input data to uppercase. You can issue the CASE M subcommand to change this setting, then reissue the FIND subcommand.

```
36   change /on a cms/about a CMS
     NOT FOUND
     = zone 1 *
37   List information about a CMS file                          LISTFILE
     top
     TOF:
38   change /cms/CMS/ *
     Alter the name of a CMS file                               RENAME
     Change the name of a CMS file                              RENAME
        .
        .
        .
     Verify the existence of a CMS file on a read/write disk    STATEW
     EOF:
39   save
     EDIT:
     top
     TOF:
     next
40   Alter the name of a CMS file                               RENAME
     $dup
     Alter the name of a CMS file                               RENAME
     change /name/filetype/
     Alter the filetype of a CMS file                             RENAME
     next
41   Change the name of a CMS file                              RENAME
     change /name/filename/
     Change the filename of a CMS file                            RENAME
     next
42   Compare the contents of CMS disk files                     COMPARE
     next
     Copy CMS files                                             COPYFILE
43   find M
     Modify a CMS file                                          EDIT
     up
     List information about a CMS file                          LISTFILE
44   i Make a copy of a CMS disk file                             COPYFILE
     top
     TOF:
```

---

36 The editor locates the line and displays it. You want to change the character string "on a cms" to "about a CMS". The editor does not find the string you specify because the zone setting for columns 60 through 80 is still in effect. You can enter the ZONE subcommand, and reissue the CHANGE subcommand, or you can enter the = (REUSE) subcommand to stack the CHANGE subcommand, and enter the ZONE subcommand to execute first.

37 The ZONE subcommand is executed, then the CHANGE subcommand. The editor displays the changed line.

38 At the top of the file, enter another global change request, to change lowercase occurrences of the string cms to uppercase. The editor displays each line changed.

39 When the EOF: message indicates that the end of the file is reached, you can save the changes made during this edit session with the SAVE subcommand before continuing.

40 Move the current line pointer to point to the first line in the file. You want to add an entry that is similar; use the $DUP edit macro to duplicate the line, then change the copy that you made of the line.

41 You can change the word name to filename in the next line also.

42 You can scan a file, a line at a time, by issuing successive NEXT subcommands.

43 To insert a line beginning with the character M, and to maintain alphabetic sequencing, use the FIND subcommand to find the first line beginning with an M. The line to be inserted begins with the characters MA, so you want to move the line pointer up.

44 You can insert a single line into a file with the INPUT subcommand. Here, the INPUT subcommand is truncated to I, so that when you space over to write the command name in the right column, you can align it (ycu only have to allow for the two character spaces use by "i ".

```
45   /COPYFILE
     Copy CMS files                                                  COPYFILE
46   n
     Create a CMS file                                               EDIT
     n
     Display the contents of a CMS file at your terminal             TYPE
     n
     Dump CMS files onto tape                                        TAPE DUMP
     n
     Erase a CMS file                                                ERASE
47   up 3
     Create a CMS file                                               EDIT
     i Delete a file from a CMS disk                                  ERASE
     file
     R;
48   type function sort a

     Alter the name of a CMS file                                    RENAME
     Alter the filetype of a CMS file                                    RENAME
     Change the filename of a CMS file                                   RENAME
        .
        .
        .
     Verify the existence of a CMS file on a read/write disk         STATEW

     R;
49   edit function sort
     zone 58
     change / // * *
     Alter the name of a CMS file                          RENAME
     Alter the filetype of a CMS file                      RENAME
     Change the filename of a CMS file                     RENAME
        .
        .
        .
     Verify the existence of a CMS file on a read/write disk  STATEW
     EOF:
50   top
     TOF:
     change //| / *
     Alter the name of a CMS file                          | RENAME
     Alter the filetype of a CMS file                      | RENAME
     Change the filename of a CMS file                     | RENAME
        .
        .
        .
     Verify the existence of a CMS file on a read/write disk  | STATEW
     EOF:
```

---

45 Move the line pointer to the top of the file and begin scanning again. A diagonal (/) is interpreted as a LOCATE subcommand.

46 The NEXT subcommand can be truncated to "N".

47 In front of the line beginning "Display", insert a line beginning with "Delete". If you want to make any other modifications, do so. Otherwise, write this file onto disk with the FILE subcommand.

48 Verify your changes.

49 Edit the file again. To compress unnecessary spaces in right hand columns, change the zone setting. This time, issue a CHANGE subcommand that will delete all blank spaces occuring after column 58. Since some changes you made to the file might have spoiled the alignment in the command column, this CHANGE subcommand should realign all of the columns.

50 Return the current line pointer to the top of the file. Change a null string to the string "| " for all lines in the file; since the left zone is still column 58, the characters are inserted in columns 58 and 59.

```
51   zone 1 *
     top
     TOF:
     c //| / *
     | Alter the name of a CMS file                              | RENAME
     | Alter the filetype of a CMS file                          | RENAME
     | Change the filename of a CMS file                         | RENAME
        .
        .
        .
     | Verify the existence of a CMS file on a read/write disk   | STATEW
     EOF:
52   top
     TOF:
     next
     | Alter the name of a CMS file                              | RENAME
     tabset 72
     repeat *
     overlay |
     | Alter the name of a CMS file                              | RENAME    |
     | Alter the filetype of a CMS file                          | RENAME    |
     | Change the filename of a CMS file                         | RENAME    |
     | Compare the contents of CMS disk files                    | COMPARE   |
        .
        .
        .
     | Verify the existence of a CMS file on a read/write disk   | STATEW    |
     EOF:
     bottom
     | Verify the existence of a CMS file on a read/write disk   | STATEW    |
53   input
54   zone 1 72
     c / /-/ 1 *
     ------------------------------------------------------------------------
     top
     TOF:
55   input
     c / /-/ 1 *
     ------------------------------------------------------------------------
56   file
     R;
     print function sort
     R;
```

---

51 Change the left zone setting to column 1 and let the right zone be equal to the record length; issue the CHANGE subcommand to insert the "| " in columns 1 and 2. CHANGE can be abbreviated as "C".

52 At the top of the file, change the TABSET subcommand setting to 72. This makes column 72 the left margin. The REPEAT * subcommand, followed by the OVERLAY subcommand, indicates that all the lines in the file are to be overlaid with a | in the leftmost column (column 72).

53 When you enter this INPUT subcommand, enter a number of blank spaces following it; this places a blank line in the file.

54 Reset the ZONE setting to columns 1 and 72. The CHANGE subcommand indicates that all blanks on this line should be changed to hyphens (-). Only the blanks within the specified zone are changed.

55 Insert another blank line at the top of the file and change it to hyphens.

56 Write the file onto disk and use the CMS PRINT command to spool a copy to the offline printer.

Sample Terminal Session Using Line-Number Editing

This terminal session shows how a terminal session using right-handed line-number editing might appear on a typewriter terminal. The commands function the same way on a display terminal, but the display is somewhat different. When you enter these input lines, you should have physical tab stops set at your terminal at positions 16 and 22 (for assembler columns 10 and 16; the difference compensates for the line numbers, as you will see). On a display terminal, tab settings have no significance; once the line is in the output display area, it has the proper number of spaces.

```
1    edit test assemble
     NEW FILE:
     EDIT:
2    linemode right
     input
     INPUT:
3    00010 * sample of linemode right
     00020 test     csect
     00030          balr  12,0
     00040          using *,12
     00050          st    14,sav14
     00060          wrterm testing...
     00070          l     14,sav14
     00080          br    14
     00090          end
     00100
4
     EDIT:
5    60
     00060          WRTERM               TESTING...
6    c /testing.../'testing...'
     00060          WRTERM               'TESTING...'
7    80
     00080          BR    14
     input
     INPUT:
```

---

1 Use the EDIT command to invoke the CMS Editor. Since this is a new file, the editor issues the NEW FILE message.

2 Issue the LINEMODE subcommand to indicate that you want to begin line-number editing. For ASSEMBLE files, you cannot have line numbers on the left, because the assembler expects data in columns 1 through 7.

3 As soon as you issue the INPUT subcommand, the editor begins prompting you to enter input lines. For convenience in entering lines, the line numbers appear on the left, as they would if you were using left-handed line-number editing. In your ASSEMBLE file, however, the line numbers are actually on the right.

4 When you are have finished entering these input lines, enter a null line to return to edit mode from input mode.

5 To locate lines when you are using line-number editing, you can enter the line number of the line. In this case, enter 60 to position the current line pointer at the line numbered 00060. The editor displays the line.

6 Issue the CHANGE subcommand to place quotation marks around the text line for the WRTERM macro. The editor redisplays the line, with the change.

7 Issue the nnnnn subcommand, specifying line number 80, and use the INPUT subcommand so you can begin entering more input lines.

```
8     00083 sav14     ds    f
      00085 wkarea    ds    3d
      00087 flag      ds    x
      00088 runon     equ   x'80'
      00089 runoff    equ   x'40'
9     RENUMBER LINES
      EDIT:
      linemode off
      serial abc
      save
10    EDIT:
11    linemode right
      type
      00130 RUNOFF    EQU   X'40'
12    verify 1 *
      type
      00130 RUNOFF    EQU   X'40'                                    ABC00130
13    135             runmix   equ x'20'
14    50
      00050            ST    14,SAV14                                ABC00050
      input
      INPUT:
      00053           tm    flag,runon
      00055           bcr   1,14
      00057

15    EDIT:
      top
      TOF:
      next
      * SAMPLE OF LINEMODE RIGHT                                    ABC00010
16    restore
```

---

8 When you begin entering input lines between two existing lines, the editor uses an algorithm to assign line numbers.

9 The editor ran out of line numbers, since the next line in the file is already numbered 90. You must renumber the lines. Before you can renumber the lines, you must turn line-number editing off. Before issuing the SAVE subcommand, which writes the file and its new line numbers onto disk, you can issue the SERIAL subcommand. SERIAL ABC indicates that you want the characters ABC to appear as the first three characters of each serial number.

10 The EDIT message indicates that the SAVE request has completed.

11 Issue the LINEMODE subcommand to restore line-number editing. Use the TYPE subcommand to verify the position of the current line pointer.

12 If you want to see the serial numbers in columns 72 through 80, issue the VERIFY subcommand, specifying *, or the record length. Normally, the editor does not display the columns containing serial numbers while you are editing.

13 You can use the nnnnn subcommand to insert individual lines of text. This subcommand inserts a line that you want numbered 135, and places it in its proper position in the file. Note that although, in this example, the current line pointer is positioned at line 130, it does not need to be at the proper place in the file. When the subcommand is complete, however, the current line pointer is positioned following the line just inserted.

14 Position the line pointer at the line numbered 50, and again begin entering the input lines indicated.

15 Enter a null line to return to edit mode, move the current line pointer to the top of the file, and display the first line.

16 The RESTORE subcommand restores the default settings of the editor, and the the verification columns are restored to 1 and 72, so that line numbers are not displayed in columns 72 through 80.

```
17   type *
     * SAMPLE OF LINEMODE RIGHT
     TEST      CSECT
               BALR  12,0
               USING *,12
               ST    14,SAV14
               TM    FLAG,RUNON
               BCR   1,14
               WRTERM  'IESTING...'
               L     14,SAV14
               BR    14
     SAV14     DS    F
     WKAREA    DS    3D
     FLAG      DS    X
     RUNON     EQU   X'80'
     RUNOFF    EQU   X'40'
     RUNMIX    EQU   X'20'
               END
     EOF:
     linemode right
     file
18   RESERIALIZATION SUPPRESSED
     R;
19   type test assemble

     * SAMPLE OF LINEMODE RIGHT                                    ABC00010
     TEST      CSECT                                              ABC00020
               BALR  12,0                                         ABC00030
               USING *,12                                         ABC00040
               ST    14,SAV14                                     ABC00050
               TM    FLAG,RUNON                                      00053
               BCR   1,14                                            00055
               WRTERM 'TESTING...'                                ABC00060
               L     14,SAV14                                     ABC00070
               BR    14                                           ABC00080
     SAV14     DS    F                                            ABC00090
     WKAREA    DS    3D                                           ABC00100
     FLAG      DS    X                                            ABC00110
     RUNON     EQU   X'80'                                        ABC00120
     RUNOFF    EQU   X'40'                                        ABC00130
     RUNMIX    EQU   X'20'                                           00135
               END                                                ABC00140
```

---

17 Use the TYPE subcommand to display the file.

18 When you issue the FILE subcommand to write the file onto disk, the editor issues the message RESERIALIZATION SUPPRESSED to indicate that it is not going to update the line numbers, so that the current line numbers match the line numbers as they existed when the SAVE subcommand was issued.

19 If you want to see how the file exists on disk, use the CMS TYPE command to display the file. Note that the lines inserted after the SAVE subcommand do not have the initial ABC characters, and that they retain the line numbers they had when they were inserted.

Sample Terminal Session For OS Programmers

The following terminal session shows how you might create an assembler language program in CMS, assemble it, correct assembler errors, and execute it. All the lines that appear in lowercase are lines that you should enter at the terminal. Uppercase data represents the system response that you should receive when you enter the command.

The input data lines in the example are aligned in the proper columns for the assembler; if you are using a typewriter terminal, you should set your terminal's tab stops at columns 10, 16, 31, 36, 41, and 46, and use the Tab key when you want to enter text in these columns. If you are using a display terminal, when you use a PF key defined as a tab, or some input character, the line image is expanded as it is placed in the screen output area.

There are some errors in the terminal session, so that you can see how to correct errors in CMS.

```
1    edit ostest assemble
     NEW FILE:
     EDIT:
     input
     INPUT:
     dataproc csect
              print nogen
              space
     r0       equ   0
     r1       equ   1
     r2       equ   2
     r10      equ   10
     r12      equ   12
     r13      equ   13
     r14      equ   14
     r15      equ   15
              space
              stm   r14,r12,12(r13)  save caller's regs
              balr  r12,0           establish
              using *,r12              addressability
              st    r13,savearea+4  store addr of caller's savearea
              la    r15,savearea    get the address of my savearea
              st    r15,8(r13)      store addr in caller's savearea
              lr    r13,r15         save addr of my savearea
              space
     *open files and check that they opened okay
              space
              la    r3,0            initially set return code
              open  (indata,outdata,(output))     open files
              using ihadcb,r10      get dsect to check files
              la    r10,indata      prepare to check output file
              tm    dcboflgs,x'10'  everything ok?
              bnz   checkout        ...continue
              la    r3,100          set return code
              b     exit            ...exit
     checkout la    r10,outdata     check output file
              tm    dcboflgs,x'10'  is it okay?
              bnz   process         ...
              la    r3,200          set return code
              b     exit
              space
     process  equ   *
              get   indata          read a record from input file
```

---

1 The EDIT command is issued to create a file named OSTEST ASSEMBLE. Since the file does not exist, the editor indicates that it is a new file and you can use the INPUT subcommand to enter input mode and begin entering the input lines.

```
            lr    r2,r1           save address of record
            put   outdata,(2)     move it to output
            b     process         continue until end-of-file
            space
exit        equ   *
            close (indata,,outdata)   close files
            l     r13,savearea+4 addr of caller's save area
            lr    r15,r3          load return code
            l     r14,12(r13)     get return address
            lm    r0,r12,20(r13)  restore regs
            br    r14             bye...
            space
savearea    dc    18f'0'
indata      dcb   ddname=indd,macrf=gl,dsorg=ps,recfm=f,lrecl=80,
2
      EDIT:
      $mark
3     save#input
      EDIT:
      INPUT:
                  eodad=exit
outdata     dcb   ddname=outdd,macrf=pm,dsorg=ps
            dcbd
            space
            end
4
      EDIT:
      file
      R;
5     global maclib osmacro
      R;
6     assemble ostest
      *
      *
      *
      *
      *
      *
      *
```

---

2 Since the DCB macro statement takes up more than one line, you have to enter a continuation character in column 7&.. To do this, you can enter a null line to return to edit mode and execute the $MARK edit macro, which places an asterisk in column 7&.. If the $MARK edit macro is not on your system, you will have to enter a continuation character some other way. (See "Entering a Continuation Character in Column 72" in "Section 5. The Editors.")

3 Before continuing to enter input lines, the EDIT subcommand SAVE is issued to write what has already been written onto disk. The CP logical line end symbol (#) separates the SAVE and INPUT subcommands.

4 A null line returns you to edit mode. You may wish, at this point, to proofread your input file before issuing the FILE subcommand to write the ASSEMBLE file onto disk.

5 Since this assembler program uses OS macros, you must issue the GLOBAL command to identify the CMS macro library, OSMACRO MACLIB, before you can invoke the assembler.

6 The ASSEMBLE command invokes the VM/SP assembler to assemble the source file; the asterisks (*) indicate the CMS blip character, which you may or may not have made active for your virtual machine.

```
7    ASSEMBLER DONE
     OST00230       23          LA    R3,0           INITIALLY SET RETURN CODE
     IFO188 R3 IS AN UNDEFINED SYMBOL
     OST00240       24          OPEN  (INDATA,OUTDATA,(OUTPUT))    OPEN FILES
      4000000       27+   12,***  IHB002  INVALID OPTION OPERAND SPECIFIED-OUTDATA
     IFO197 *** MNOTE ***
     OST00290       32          LA    R3,100         SET RETURN CODE
     IFO188 R3 IS AN UNDEFINED SYMBOL
     OST00340       37          LA    R3,200         SET RETURN CODE
     IFO188 R3 IS AN UNDEFINED SYMBOL
     OST00460       63          LR    R15,R3         LOAD RETURN CODE
     IFO188 R3 IS AN UNDEFINED SYMBOL
     NUMBER OF STATEMENTS FLAGGED IN THIS ASSEMBLY =     5
     R(00012);
8    edit ostest assemble
     locate /r2
     R2       EQU   2
     i r3       equ   3
     /open
              OPEN   (INDATA,OUTDATA,(OUTPUT))    OPEN FILES
     c /,/,,/
              OPEN   (INDATA,,OUTDATA,(OUTPUT))    OPEN FILES
9    file
     R;
     assemble ostest
     *
     *
     *
     *
     *
     *

10   ASSEMBLER DONE
     NO STATEMENTS FLAGGED IN THIS ASSEMBLY
     R;
11   filedef indd disk test data a
     R;
12   filedef outdd punch
     R;
13   #cp spool punch to *
14   load ostest
```

---

7 The assembler displays errors encountered during assembly. Depending on how accurately you copied the program in this sample session, you may or may not receive some of these messages; you may also have received additional messages.

8 You must edit the file OSTEST ASSEMBLE and correct any errors in it. The errors placed in the example included a missing comma on the OPEN macro, and the omission of an EQU statement for a general register. These changes are made as shown. The CMS Editor accepts a diagonal (/) as a LOCATE subcommand.

9 After all the changes have been made to the ASSEMBLE file, you can issue the FILE subcommand to replace the existing copy on disk, and then reassemble it.

10 This time, the assembler completes without encountering any errors. If your ASSEMBLE file still has errors, you should use the editor to correct them.

11 The FILEDEF command is used to define the input and output files used in this program. The ddnames INDD and OUTDD, defined in the DCBs in the program, must have a file definition in CMS. To execute this program, you should have a file on your A-disk name TEST DATA, which must have fixed-length, 80-character records. If you have no such file, you can make a copy of your ASSEMBLE file as follows:

```
copyfile ostest assemble a test data a
```

12 The output file is defined as a punch file, so that it will be written to your virtual card punch.

13 The CP SPOOL command is issued, using the #CP function, to spool your virtual punch to your virtual card reader. When you use the #CP function, you do not receive a Ready message.

14 The LOAD command loads the TEXT file produced by the assembly into virtual storage. The START command begins program execution.

```
    R;
    start
    DMSLIO740I EXECUTION BEGINS...
15  DMSSOP036E OPEN ERROR CODE '04' ON 'OUTDD '.
    R(00200);
16  filedef
    INDD     DISK     TEST     DATA     A1
    OUTDD    PUNCH
    R;
17  filedef outdd punch (lrecl 80 recfm f
    R;
18  #cp query reader all
    NO RDR FILES
19  load ostest (start
    DMSLIO740I EXECUTION BEGINS...
20  PUN FILE 6198  TO  BILBO    COPY 01 NOHOLD
    R;
21  fi indd reader
    R;
    fi outdd disk new osfile a4 (recfm fb block 1600 lrecl 80
    R;
22  listfile new osfile a4 (label
    DMSLST002E FILE NOT FOUND.
    R(00028);
23  run ostest
    EXECUTION BEGINS...
    *
    R;
24  listfile new osfile a4 (label
    FILENAME FILETYPE  FM  FORMAT  LRECL  RECS BLOCKS   DATE     TIME     LABEL
    NEW      OSFILE    A4  F        1600     5     10  9/30/75  8:26:14  PAT198
    R;
```

---

15 An open error is encountered during program execution. The CMS ready message indicates a return code of 200, which is the value placed in it by your program.

16 The FILEDEF command, with no operands, results in a display of the current file definitions in effect.

17 Error code 4 on an open request means that no RECFM or LRECL information is available. An examination of the program listing would reveal that the DCB for OUTDD does not contain any information about the file format; you must supply it on the FILEDEF command. Re-enter the FILEDEF command.

18 You can use the CP QUERY command to determine whether there are any files in your card reader. It should be empty; if not, determine whether they might be files you need, and if so, read them into your virtual machine; otherwise, purge them.

19 Use the LOAD command to execute the program again; this time, use the START option of the LOAD command to begin the program execution.

20 The PUN FILE message indicates that a file has been transferred to your virtual card reader. The ready message indicates that your program executed successfully.

21 For the next execution of this program, you are going to read the file back out of your card reader and create a new CMS disk file, in OS simulated data set format. FI is an acceptable system truncation for the command name, FILEDEF.

22 The LISTFILE command is issued to check that the file NEW OSFILE does not exist.

23 The RUN command (which is an EXEC procedure) is used instead of the LOAD and START commands, to load and execute the program. The ready message indicates that the program completed execution.

24 The LISTFILE command is issued again, and the file NEW OSFILE is listed. (If you issue another CP QUERY READER command, you will also see that the file is no longer in your card reader.)

Sample Terminal Session for DOS Programmers

The following terminal session shows how you might create an assembler language program in CMS, assemble it, correct assembler errors, and execute it. All the lines that appear in lowercase are lines that you should enter at the terminal. Uppercase data represents the system response that you should receive when you enter the command.

The input data lines in the example are aligned in the proper columns for the assembler; if you are using a typewriter terminal, you should set your terminal's tab stops at columns 10, 16, 31, 36, 41, and 46 and use the Tab key when you want to enter text in these columns. If you are using a display terminal, when you use a PF key or an input character defined as a tab, the line image is expanded as it is placed in the screen output area.

Note: The assembler, in CMS, cannot read macros from VSE/AF libraries. This sample terminal session shows how to copy macros from VSE/AF libraries and create CMS MACLIB files. Ordinarily, the macros you need should already be available in a system MACLIB file. You do not have to create a MACLIB each time you want to assemble a program.

There are some errors in the terminal session, so that you can see how to correct errors in CMS.

```
1    cp link dosres 130 130 rr linkdos
     DASD 130 LINKED R/O
     R;
     access 130 z
     Z (130) R/O - DOS
     R;
2    set dos on z
     R;
3    edit dostest assemble
     NEW FILE:
     EDIT:
     input
     INPUT:
     begpgm   csect
              balr  12,0
              using *,12
              la    13,savearea
              open  infile,outfile
     loop     get   infile
              put   outfile
              b     loop
     eodad    equ   *
              close infile,outfile
              eoj
              eject
     buffer   dc    CL80' '
     infile   dtfdi modname=shrmod,ioarea1=buffer,devaddr=sysipt,
4
     EDIT:
```

---

1 Use the CP LINK command to link to the DOS system residence volume and the ACCESS command to access it. In this example, the system residence is at virtual address 130 and is accessed as the Z-disk.

2 Enter the CMS/DOS environment, specifying the mode letter at which the DOS/VS (VSE/AF) system residence is accessed.

3 Use the EDIT command to create a file named DOSTEST ASSEMBLE. Since the file does not exist, the editor indicates that it is a new file and you can use the INPUT subcommand to enter input mode and begin entering the input lines.

4 Since the DTFDI macro statement takes up more than one line, you have to enter a continuation character in column 72. To do this, you can enter a null line to return to edit mode and execute the $MARK edit macro, which places an asterisk in column 72. If the $MARK edit macro is not on your system, you will have to enter a continuation character some other way. (See "Entering a Continuation Character in Column 72" in "Section 5. The Editors.")

```
     $mark
5    save#input
     EDIT:
     INPUT:
                    eofaddr=eodad,recsize=80
     outfile  dtfdi modname=shrmod,ioarea1=buffer,devaddr=syspch,
6
     EDIT:
     $mark
     save#input
     EDIT:
     INPUT:
                     recsize=81
     shrmod   dimod typefle=output
     endpgm   equ   *
              end
7

     EDIT:
     file
     R;
8    edit getmacs eserv
     NEW FILE:
     EDIT:
     tabs 2 72
     input
     INPUT:
9    punch open,close,get,put,dimod,dtfdi

     EDIT:
     file
     R;
10   assgn sysipt a
     R;
     eserv getmacs
     R;
```

---

5 Before continuing to enter input lines, the EDIT subcommand SAVE is issued to write what has already been written onto disk. The CP logical line end symbol (#) separates the SAVE and INPUT subcommands.

6 Another continuation character is needed.

7 A null line returns you to edit mode. You may want, at this point, to proofread your input file before issuing the FILE subcommand to write the ASSEMBLE file on disk.

8 To obtain the macros you need to assemble this file, use the editor to create an ESERV file. By setting the logical tabs at columns 2 and 72, you can protect yourself from entering data in column 1.

9 PUNCH is an ESERV program control statement that copies and de-edits macros from source statement libraries; in this case, the system source statement library. The output is directed to the SYSPCH device, which the CMS/DOS ESERV EXEC assigns by default to your A-disk.

10 You must assign the logical unit SYSIPT before you invoke the ESERV command. GETMACS is the filename of the ESERV file containing the ESERV control statements.

```
11   listfile getmacs *
     GETMACS  ESERV    A1
     GETMACS  MACRO    A1
     GETMACS  LISTING  A1
     R;
12   maclib gen dosmac getmacs
     R;
     erase getmacs *
     R;
13   global maclib dosmac
     R;
14   assemble dostest
     *
     *

15   ASSEMBLER DONE
     DOS00040        4          LA    13,SAVEAREA
     IFO188 SAVEAREA IS AN UNDEFINED SYMBOL
     DOS00110       35          EOJ
     IFO078 UNDEFINED OP CODE
     NUMBER OF STATEMENTS FLAGGED IN THIS ASSEMBLY =     2
     R(00008);
16   edit dotest assemble
     EDIT:
     locate /buffer/
     BUFFER    DC    CL80' '
     input savearea  ds    9d
     file
     R;
17   edit eoj eserv
     NEW FILE:
     EDIT:
     i   punch eoj
     file
     R;
18   listio sysipt
      SYSIPT  DISK      A
     R;
     eserv eoj
     R;
```

---

11 After the ESERV EXEC completes execution, you have three files. You may want to examine the LISTING file to check the ESERV program listing. The MACRO file contains the punch (SYSPCH) output.

12 The MACLIB command creates a macro library named DOSMAC MACLIB. Since the MACLIB command completed successfully, you can erase the files GETMACS ESERV, GETMACS LISTING, and GETMACS MACRO; an asterisk in the filetype field of the ERASE command indicates that all files with the filename of GETMACS should be erased.

13 Before you can invoke the assembler, you have to identify the macro library that contains the macros; use the GLOBAL command, specifying DOSMAC MACLIB.

14 The ASSEMBLE command invokes the VM/SP assembler to assemble the source file; the asterisks (*) indicate the CMS blip character, which you may or may not have made active for your virtual machine.

15 The assembler displays errors encountered during assembly. Depending on how accurately you copied the program in this sample session, you may or may not receive some of these messages; you may also have received additional messages.

16 To correct the first error, which was the omission of a DS statement for SAVEAREA, edit the file DOSTEST ASSEMBLE and insert the missing line.

17 The second error indicates that the macro EOJ is not available, since it was not copied from the source statement library. Create another ESERV file to punch this macro.

18 Use the LISTIO command to check that SYSIPT is still assigned to your A-disk, so that you do not have to issue the ASSGN command again. Then issue the ESERV command again, this time specifying the filename EOJ.

```
19   maclib add dosmac eoj
     R;
     maclib map dosmac (term
     MACRO    INDEX  SIZE
     OPEN         2    43
     CLOSE       46    43
     GET         90    56
     PUT        147    93
     DIMOD      241   647
     DTFDI      889   284
     EOJ       1174     6
     R;
20   erase eoj *
     R;
     assemble dostest
     *
     *
     *

21   ASSEMBLER DONE
     NO STATEMENTS FLAGGED IN THIS ASSEMBLY
     R;
22   listfile dostest *
     DOSTEST  ASSEMBLE A1
     DOSTEST  LISTING  A1
     DOSTEST  TEXT     A1
     R;
     print dostest listing
     R;
23   doslked dostest
     R;
24   listfile dostest *
     DOSTEST  ASSEMBLE A1
     DOSTEST  DOSLIB   A1
     DOSTEST  TEXT     A1
     DOSTEST  LISTING  A1
     DOSTEST  MAP      A5
     R;
```

---

19 Use the ADD function of the MACLIB command to add the macro EOJ to DOSMAC MACLIB. Then, issue the MACLIB command again, using the MAP function and the TERM option to display a list of the macros in the library.

20 Erase the EOJ files. You should always remember to erase files that you do not need any longer. Reassemble the program.

21 This time, the assembler completes without encountering any errors. If your ASSEMBLE file still has errors, you should use the editor to correct them.

22 Use the LISTFILE command to check for DOSTEST files. The assembler created the files, DOSTEST LISTING and DOSTEST TEXT. The TEXT file contains the object module. You can print the program listing, if you want a printed copy. Then, you may want to erase it.

23 Use the DOSLKED command to link-edit the TEXT file into an executable phase and write it into a DOSLIB. Since this program has no external references, you do not need to add any linkage editor control statements.

24 Now, you have a DOSTEST DOSLIB, containing the link-edited phase, and a MAP file, containing the linkage editor map. You can display the linkage editor map with the TYPE command, or use the PRINT command if you want a printed copy.

```
25   #cp spool punch to *
     punch test data a
     PUN FILE 0100  TO BILBO    COPY 01 NOHOLD
     R;
     #cp query reader all
     ORIGINID FILE CLASS RECDS  CPY HOLD DATE  TIME     NAME        TYPE     DIST
     PATTI    5840 A PUN 000097 01  NONE 09/29 15:00:39 TEST        DATA     BIN211
26   assgn sysipt reader
     R;
     assgn syspch a
     R;
27   dlbl outfile a cms punch output (syspch
     R;
     state punch output a
     DMSSTT002E FILE NOT FOUND.
     R(00028);
28   global doslib dostest
     R;
     fetch dostest
     DMSFET710I PHASE 'DOSTEST' ENTRY POINT AT LOCATION 020000.
     R;
29   start
     DMSLIO740I EXECUTION BEGINS...
     R;
     listfile punch output a (label
     FILENAME FILETYPE  FM  FORMAT  LRECL  RECS BLOCKS   DATE     TIME     LABEL
     PUNCH    OUTPUT    A1  F          80    97     10  9/29/79 14:50:55  BBB191
     R;
     #cp query reader all
     NO RDR FILES
```

---

25 To execute this program in CMS/DOS, punch a file that has fixed-length 80-character records into your virtual card punch. If you do not have any files that have fixed-length, 80-character records, you can create a file named TEST DATA with the CMS Editor, or by copying your ASSEMBLE source file with the COPYFILE command, as follows:

```
copyfile dostest assemble a test data a
```

Use the CP SPOOL command to spool the punch to your own virtual machine, then use the PUNCH command to punch the file. The PUN FILE message indicates that the file is in your card reader. Use the CP QUERY command to check that it is the first, or only file in your reader.

26 Use the ASSGN command to assign SYSIPT to your card reader and SYSPCH to your A-disk.

27 When you assign a logical unit to a disk mode, you must issue the DLBL command to identify the disk file to CMS. For this program execution, you are creating a CMS file named PUNCH OUTPUT. The STATE command ensures that the file does not already exist. If it does exist, rename it, or else use another filename or filetype on the DLBL command.

28 Use the GLOBAL command to identify the DOSLIB, DOSTEST, you want to search for executable phases, then issue the FETCH command specifying the phase name. The FETCH command loads the executable phase into storage. When the FETCH command is executed without the START option, a message is displayed indicating the entry point location of the program loaded.

29 The START command begins program execution. The CMS ready message indicates that your program completed successfully. You can check the input and output activity by using the LISTFILE command to list the file PUNCH OUTPUT. If you use the CP QUERY command, you can see that the file is no longer in your virtual card reader.

```
30  assgn sysipt a
    R;
    dlbl infile a cms punch output (sysipt
    R;
    assgn syspch punch
    R;
31  fetch dostest (start
    DMSLIO740I EXECUTION BEGINS...
32  PUN FILE 5829  TO  BILBO    COPY 01 NOHOLD
    R;
    read punch2 output
    R;
    listfile punch2 output a (label
    FILENAME FILETYPE  FM  FORMAT  LRECL  RECS BLOCKS   DATE     TIME     LABEL
    PUNCH2   OUTPUT    A1  F         80    97     10  9/29/75 14:50:59  BBB191
    R;
```

---

30 If you want to execute this program again, you can assign SYSIPT and SYSPCH to different devices; in this example, the input disk file PUNCH OUTPUT is written to the virtual punch. You do not need to reissue the GLOBAL DOSLIB command; it remains in effect until you reissue it or IPL CMS again.

31 This time, the program execution starts immediately, because the START option is specified on the FETCH command line.

32 Again, the PUN FILE message indicates that a file has been received in your virtual card reader. You can use the CMS command READCARD to read it onto disk and assign it a filename and filetype, in this example, PUNCH2 OUTPUT.

Sample Terminal Session Using Access Method Services

This sample terminal session shows you how to use access method services under CMS. You should have an understanding of VSAM and access method services before you use this terminal session.

The terminal session uses a number of CMS files, which you may create during the course of the terminal session; or, you may prefer to create all of the files that you need beforehand. Within the sample terminal session, the file that you should create is displayed prior to the commands that use it.

This terminal session is for both CMS OS VSAM programmers and CMS/DOS VSAM programmers; all the ASSGN commands and SYSxxx operands that apply when the CMS/DOS environment is active are shaded. If you have issued the command SET DOS ON, you must enter the shaded entries; if not, you must omit the shaded entries.

Notes:

1. This terminal session assumes that you have, to begin with, a read/write CMS A-disk. This is the only disk required. Additional disks used in this exercise are temporary disks, formatted with the Device Support Facility program under CMS. If you have OS or DOS disks available, you should use them, and remember to supply the proper volume and virtual device address information, where appropriate. The number of cylinders available to users for temporary disk space varies among installations; if you cannot acquire ample disk space, see your system support personnel for assistance.

2. Output listings created by AMSERV take up disk space, so if your A-disk does not have a lot of space on it, you may want to erase the LISTING files created after each AMSERV step.

3. If any of the AMSERV commands that you execute during this sample terminal session issue a nonzero return code; for example:

   ```
   R(00012);
   ```

   You should edit the LISTING file to examine the access method services error messages. The publication VSE/VSAM Messages and Codes contains the return codes and reason codes issued by access method services. You should determine the cause of the error, examine the DLBL commands and AMSERV files you used, correct any errors, and retry the command.

```
1   #cp define t3330 200 10
    DASD 200 DEFINED
    #cp query virtual 200
    DASD 200 3330 (TEMP) R/W      10 CYL

    #cp define t3330 300 10
    DASD 300 DEFINED
    #cp query virtual 300
    DASD 300 3330 (TEMP) R/W      10 CYL

    #cp define t3330 400 10
    DASD 400 DEFINED
    #cp query virtual 400
    DASD 400 3330 (TEMP) R/W      10 CYL
```

---

1 These commands define temporary 3330 mindisks at virtual addresses 200, 300, and 400.

```
2    File: PUNCH DSF

      INIT UNIT(200) DEVTYP(3330) NVFY VOLID(222222) DVTOC(0,1,1) -
        MIMIC(MINI(10))
      INIT UNIT(300) DEVTYP(3330) NVFY VOLID(333333) DVTOC(0,1,1) -
        MIMIC(MINI(10))
      INIT UNIT(400) DEVTYP(3330) NVFY VOLID(444444) DVTOC(0,1,1) -
        MIMIC(MINI(10))


3    File: DSF EXEC

     &TRACE OFF
     &CNTRL = &1
     &COMMAND CP CLOSE READER
     &COMMAND CP PURGE READER CLASS I
     &COMMAND CP SPOOL PUNCH CONT TO * CLASS I
     &COMMAND PUNCH IPL DSF S ( NOH
     &COMMAND PUNCH &CNTRL &FILENAME ( NOH
     &COMMAND CP SPOOL PUNCH NOCONT CLOSE
     &COMMAND CP SPOOL READER CLASS I NOHOLD
     &COMMAND CP IPL 00C CLEAR ATTN


4    exec dsf punch

     NO FILES PURGED
     PUN FILE nnnn  TO  CAMPBEL COPY 001   NOHOLD


5    ICK005E DEFINE INPUT DEVICE, REPLY 'DDDD,CUU' or 'CONSOLE'
     ENTER INPUT/COMMAND:
```

---

2 This file contains control statements for the Device Support Facility program, which initializes disks for use by VSAM. These disks are labelled 222222, 333333, and 444444.

3 This file contains the commands necessary to use the Device Support Facility program in a virtual machine.

4 Execute the DSF EXEC, specifying that the Device Support Facility control statements contained in the file 'PUNCH DSF' should be appended to the standalone Device Support Facility program.

5 These messages are issued by the Device Support Facility standalone program.

```
6    2540,00c
     2540,00C
     ICK006E DEFINE OUTPUT DEVICE, REPLY 'DDDD,CUU' or 'CONSOLE'
     ENTER INPUT/COMMAND:


7    console
     CONSOLE
     ICKDSF -  SA DEVICE SUPPORT FACILITIES 5.0  TIME20:26:00  03/09/82   PAGE  1
      INIT UNIT(200) DEVTYP(3330) NVFY VOLID(222222) DVTOC(0,1,1) -
        MIMIC(MINI(10))
     ICK00700I 200 BEING PROCESSED AS LOGICAL DEVICE = 3330     PHYSICAL DEVICE = 3330-11
     ICK003D REPLY U TO ALTER VOLUME 200 CONTENTS, ELSE T
     ENTER INPUT/COMMAND:


8    u
     U
     ICK01314I VTOC IS LOCATED AT CCHH=X'0000 0001' AND IS     1 TRACKS.
     ICK00001I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS 0

      INIT UNIT(300) DEVTYP(3330) NVFY VOLID(333333) DVTOC(0,1,1) -
        MIMIC(MINI(10))
     ICK00700I 300 BEING PROCESSED AS LOGICAL DEVICE = 3330     PHYSICAL DEVICE = 3330-11

     ICK003D REPLY U TO ALTER VOLUME 300 CONTENTS, ELSE T
     ENTER INPUT/COMMAND:
     u
     U
     ICK01314I VTOC IS LOCATED AT CCHH=X'0000 0001' AND IS     1 TRACKS.
     ICK00001I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS 0

      INIT UNIT(400) DEVTYP(3330) NVFY VOLID(444444) DVTOC(0,1,1) -
        MIMIC(MINI(10))
     ICK00700I 400 BEING PROCESSED AS LOGICAL DEVICE = 3330     PHYSICAL DEVICE = 3330-11

     ICK003D REPLY U TO ALTER VOLUME 400 CONTENTS, ELSE T
     ENTER INPUT/COMMAND:
     u
     U
     ICK01314I VTOC IS LOCATED AT CCHH=X'0000 0001' AND IS     1 TRACKS.
     ICK00001I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS 0

     ICKDSF MAXIMUM STORAGE USED = 278968 BYTES  (FIXED = 258120, DYNAMIC = 020848)
     ICK00002I ICKDSF PROCESSING COMPLETE. MAXIMUM CONDITION CODE WAS 0
```

---

6 Since the Device Support Facility control statements reside in the virtual card reader, you must indicate to Device Support Facility the device type and the address of your virtual reader.

7 This response tells Device Support Facility to output all run time information to your virtual machine console.

8 This response gives Device Support Facility permission to proceed with the initialization of the disk.

```
9     #cp ipl cms parm autocr
      CMS VM/SP1.2.0 SL 000
      R;

10    #cp link vseaf 350 350 rr pass=read
      DASD 350 LINKED R/O; R/W BY GANDALF
      access 350 z
      DMSACC723I Z (350) R/O - DOS
      R;
      set dos on z ( vsam
      R;

11    access 200 b
      DMSACC723I B (200) R/W - DOS
      R;
      access 300 c
      DMSACC723I C (300) R/W - DOS
      R;
      access 400 d
      DMSACC723I D (400) R/W - DOS
      R;

12    query search
      PLC191  191  A    R/W
      222222  200  B    R/W - DOS
      333333  300  C    R/W - DOS
      444444  400  D    R/W - DOS
      MNT190  190  S    R/O
      MNT191  190  Y/S  R/O
      VSERES  350  Z    R/O - DOS
      R;
```

---

9 You must re-IPL CMS after all Device Support Facility processing has completed.

10 If you are a CMS/DOS user, you must access the VSE/AF SYSRES disk and issue the 'SET DOS ON fm (VSAM' command. If you have not previously linked to the VSE/AF SYSRES, you must use the CP LINK command before you issue the ACCESS command. Another method is to have the operator ATTACH the SYSRES disk to you virtual machine. Consult with your system programmer for the procedure to use at your installation.

11 ACCESS the three newly formatted disks as your B-, C-, and D-disks.

12 You can issue the QUERY SEARCH command to verify the status of all disks you currently have accessed.

```
13   File: MASTCAT AMSERV
      DEFINE MASTERCATALOG -
        (  NAME  (MASTCAT)  -
           VOLUME (222222) -
           CYL (4) -
           UPDATEPW (GAZELLE) -
           FILE (IJSYSCT) ) DATA (CYL(1))

14   assgn syscat b
     R;
     dlbl ijsysct b dsn mastcat (syscat perm extent
     DMSDLB331R ENTER EXTENT SPECIFICATIONS:
     19 171

15
     R;

16   amserv mastcat
     R;



17   File: CLUSTER AMSERV
      DEFINE CLUSTER ( NAME (BOOK.LIST  ) -
         VOLUMES (222222) -
         TRACKS (20) -
         KEYS (14,0) -
         RECORDSIZE (120,132) ) -
         DATA (NAME (BOOK.LIST.DATA) ) -
         INDEX (NAME (BOOK.LIST.INDEX ) )


18   amserv cluster
     4221A ATTEMPT 1 OF 2. ENTER PASSWORD FOR JOB AMSERV   FILE MASTCAT
     gazelle
     R;


19   File: REPRO AMSERV
       REPRO INFILE (BFILE -
             ENV ( RECORDFORMAT(F) -
             BLOCKSIZE(120) -
             PDEV (3330) ) ) -
             OUTFILE (BOOK)
```

---

13 The file MASTCAT AMSERV defines the VSAM master catalog that you are going to use and provides space for suballocated clusters.

14 Identify the master catalog volume, and use the EXTENT option on the DLBL command so that you can enter the extents. For this extent, specify 171 tracks (9 cylinders) for the master catalog. Since 4 cylinders are specified in the AMSERV file, the remaining 5 cylinders will be used for suballocation by VSAM.

15 You must enter a null line to indicate that you have finished entering extent information.

16 Issue the AMSERV command, specifying the MASTCAT file. The ready message indicates that the master catalog is created.

17 Define a suballocated cluster. This cluster is for a key-sequenced data set, named BOOK.LIST.

18 No DLBL command is necessary when you define a suballocated cluster. Note that since the password was not provided in the AMSERV file, access method services prompts you to enter the password of the catalog, which is defined as GAZELLE.

19 Use the access method services REPRO command to copy a CMS data file into the cluster that you just defined.

```
20   assgn sys001 a
     R;
     copyfile test data a (recfm f lrecl 120
     R;
     sort test data a book file a
     DMSSRT604R ENTER SORT FIELDS:
     1 14
     R;
     dlbl bfile a cms book file (sys001
     R;
21   assgn sys002 b
     R;
     dlbl book b dsn book.list (vsam sys002
     R;
     amserv repro
     R;



22   File: SPACE AMSERV
       DEFINE SPACE -
            ( FILE (SPACE) -
            TRACKS (57) -
            VOLUME (333333) )

     assgn sys003 c
     R;
23   dlbl space c (extent sys003
     DMSDLB331R ENTER EXTENT SPECIFICATIONS:
     19 57

     R;
24   amserv space
     4221A ATTEMPT 1 OF 2. ENTER PASSWORD FOR JOB AMSERV   FILE MASTCAT
     gazelle
     R;
```

---

20 You must identify the ddnames for the input and output files for the REPRO function. BFILE is a CMS file, which must be a fixed-length, 120-character file, and it must be sorted alphamerically in columns 1 through 14. The COPYFILE command can copy any existing file that you have to the proper record format; the SORT command sorts the records on the proper fields.

21 The output file is the VSAM cluster, so you must use the VSAM option on this DLBL command.

22 Create an AMSERV file to define additional space for the master catalog on the volume labelled 333333.

23 Again, use the EXTENT option on the DLBL command so that you can enter extent information, and a null line to indicate that you have finished entering extents.

24 Issue the AMSERV command. Again, you are prompted to enter the password of the master catalog.

```
25  File: UNIQUE AMSERV
     DEFINE  CLUSTER -
       ( NAME (UNIQUE.FILE) -
         UNIQUE ) -
       DATA  -
       ( CYL (3) -
         FILE (KDATA) -
         RECORDSIZE (100 132) -
         KEYS(12,0) -
         VOLUMES (333333 ) ) -
       INDEX -
       ( CYL  (1) -
         FILE (KINDEX) -
         VOLUMES (333333) )
26  dlbl kdata c (extent sys003
    DMSDLB331R ENTER EXTENT SPECIFICATIONS:
    76 57

    R;
    dlbl kindex c (extent sys003
    DMSDLB331R ENTER EXTENT SPECIFICATIONS:
    76 76

    R;
    amserv unique
    4221A ATTEMPT 1 OF 2. ENTER PASSWORD FOR JOB AMSERV   FILE MASTCAT
    gazelle
    R;
27  File: USERCAT AMSERV
     DEFINE USERCATALOG -
          ( CYL (8) -
            FILE (IJSYSUC) -
            NAME (PRIVATE.CATALOG) -
            VOLUME (444444) -
            UPDATEPW (UNICORN) -
            ATTEMPTS (2) ) -
           DATA  (CYL (3)  ) -
           INDEX ( CYL (1) ) -
           CATALOG (MASTCAT/GAZELLE )


28  assgn sys006 d
    R;
    dlbl ijsysuc d dsn private.catalog (extent sys006 perm
    DMSDLB331R ENTER EXTENT SPECIFICATIONS:
    19 152

    R;
    amserv usercat
    *
    R;



29  TAPE 181 ATTACHED
```

---

25 This AMSERV file defines a unique cluster, with data and index components.

26 You must enter DLBL commands and extent information for both the data and index components of the unique cluster.

27 Next, define a private (user) catalog for the volume 444444. This catalog is named PRIVATE.CATALOG and has a password of UNICORN. Again, as in step 13, space is made available for suballocation.

28 When you define a user catalog that you are going to use as the job catalog for a terminal session, you should use the ddname IJSYSUC.

29 You may want to try an EXPORT/IMPORT function, if you can obtain a scratch tape from the operator. When the tape is attached to your virtual machine, you receive this message.

```
30   File: EXPORT AMSERV
       EXPORT BOOK.LIST  -
          INFILE (BOOK) -
          OUTFILE (TEMP  ENV (PDEV (2400)  REWIND NOLABEL ) )
31   dlbl book b dsn book list (cat ijsysct sys002
     R;
32   amserv export (tapout 181
     DMSAMS361R ENTER TAPE OUTPUT DDNAMES:
     temp
     R;



33   File: IMPORT AMSERV

      IMPORT  -
              CATALOG (PRIVATE.CATALOG/UNICORN) -
              INFILE (TEMP ENV (PDEV (2400) REWIND NOLABEL)) -
              OBJECTS (BOOK.LIST VOL (444444))


34   amserv import (tapin 181
     DMSAMS361R ENTER TAPE INPUT DDNAMES:
     temp
     R;
```

---

30 The file that is being exported is the cluster BOOK.LIST created above. If you do not have access to a tape, you can export the file to your CMS A-disk. Remember to change the PDEV parameter to reflect the appropriate device type.

31 You must reissue the DLBL for BOOK.LIST, because there is a job catalog in effect, and the file is cataloged in the master catalog. Use the CAT option to override the job catalog.

32 There is no default tape value when you are using tapes with the AMSERV command. You must specify the TAPIN or TAPOUT option and indicate the virtual address of the tape. You are prompted to enter the ddname, which for this file is TEMP.

33 The last AMSERV file imports the cluster BOOK.LIST to the user catalog, PRIVATE.CATALOG.

34 Read the tape in as input.

Glossary

This section explains some of the terms and acronyms that appear in this manual. For a complete list of terms used in VM/SP refer to the *VM/SP Library Guide and Master Index*.

ADCON. Is an A-type address constant used in the calculation of storage addresses.

CMS (Conversational Monitor System). A component of Virtual Machine/System Product (VM/SP) that is a conversational operating system designed to run under Control Program (CP).

console stack. Refers collectively to the program stack and the terminal input buffer.

CP (Control Program). A component of Virtual Machine/System Product (VM/SP) that controls the resources of the real machine.

discontiguous saved segment. An area of storage beyond the address of your virtual machine address space (not contiguous with your virtual storage) where segments are loaded as needed.

ECB. Event Control Block

extended PLIST (untokenized parameter list). Consists of four addresses that indicate the extended form of the command as it was entered at the terminal.

IPL. Initial program load.

look-aside entry. A nucleus resident routine becomes a look-aside entry after it has been executed.

module. A nonrelocatable file whose external references have been resolved.

nucleus. That part of CP or CMS that is resident in main storage.

program stack. Temporary storage for lines (or files) being exchanged by programs that execute in CMS.

PLIST. Parameter list

REXX language. Restructured Extended Executor language used in System Product interpreter programs.

SID code. Support Identification code

terminal input buffer. Holds lines entered at your terminal until CMS processes them.

tokenized PLIST (parameter list). A string of doubleword aligned parameters occupying successive double words.

virtual machine. A functional equivalent of a real machine.

VM/SP (Virtual Machine/System Product). A program product that controls virtual machines.

Index

Special Characters

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

SC19-6210-2

VM/SP CMS User's Guide (File No. S370/4300-39) Printed in U.S.A. SC19-6210-2

Virtual Machine/System Product
CMS User's Guide
Order No. SC19-6210-2

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate. Comments may be written in your own language; English is not required.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

	Yes	*No*
• Does the publication meet your needs?	☐	☐
• Did you find the material:		
Easy to read and understand?	☐	☐
Organized for convenient use?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐

• What is your occupation? _____

• How do you use this publication:

As an introduction to the subject?	☐	As an instructor in class?	☐
For advanced knowledge of the subject?	☐	As a student in class?	☐
To learn about operating procedures?	☐	As a reference manual?	☐

Your comments:

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

If you would like a reply, please supply your name and address on the reverse side of this form.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

Reader's Comment Form

Cut or Fold Along Line

Fold and Tape | Please Do Not Staple | Fold and Tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department G60
P. O. Box 6
Endicott, New York 13760

Fold | Fold

If you would like a reply, *please print:*

Your Name _____

Company Name _____ *Department* _____

Street Address _____

City _____

State _____ *Zip Code* _____

IBM Branch Office serving you _____

IBM®

VM/SP CMS User's Guide (File No. S370/4300-39) Printed in U.S.A. SC19-6210-2

Virtual Machine/System Product
CMS User's Guide
Order No. SC19-6210-2

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate. Comments may be written in your own language; English is not required.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

	Yes	*No*
• Does the publication meet your needs?	☐	☐
• Did you find the material:		
Easy to read and understand?	☐	☐
Organized for convenient use?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐

• What is your occupation? _____

• How do you use this publication:

As an introduction to the subject?	☐	As an instructor in class?	☐
For advanced knowledge of the subject?	☐	As a student in class?	☐
To learn about operating procedures?	☐	As a reference manual?	☐

Your comments:

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

If you would like a reply, please supply your name and address on the reverse side of this form.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

Reader's Comment Form

Cut or Fold Along Line

Fold and Tape | Please Do Not Staple | Fold and Tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department G60
P. O. Box 6
Endicott, New York 13760

Fold | Fold

If you would like a reply, *please print:*

Your Name _____

Company Name _____ *Department* _____

Street Address _____

City _____

State _____ *Zip Code* _____

IBM Branch Office serving you _____

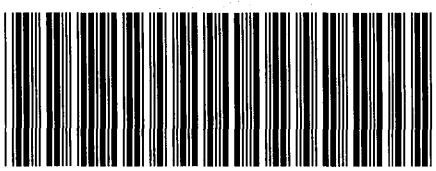
IBM®

VM/SP CMS User's Guide (File No. S370/4300-39) Printed in U.S.A. SC19-6210-2

SC19-6210-2

VM/SP CMS User's Guide (File No. S370/4300-39) Printed in U.S.A. SC19-6210-2

SC19-6210-2



SC19-6210-2

VM/SP CMS User's Guide (File No. S370/4300-39) Printed in U.S.A. SC19-6210-2